

```
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAAAAAAAAA      TTTTTTTTTTTTTTTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAAAAAAAAA      TTTTTTTTTTTTTTTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAAAAAAAAA      TTTTTTTTTTTTTTTT
EEE      MMMMMM      MMMMMM      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMMMMM      MMMMMM      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMMMMM      MMMMMM      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUUUUUUUUUUUUUUUU LLLLLLLLLLLLLLLLLL AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUUUUUUUUUUUUUUUU LLLLLLLLLLLLLLLLLL AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUUUUUUUUUUUUUUUU LLLLLLLLLLLLLLLLLL AAA      AAA      TTT
```

[illegible]

(2)	89	Miscellaneous Notes
(3)	146	DECLARATIONS
(4)	190	Conditional Assembly Parameters
(5)	225	VAX\$MOVTC - Move Translated Characters
(6)	455	VAX\$MOVTC - Move Translated Until Character
(7)	683	VAX\$CMPC3 - Compare Characters (3 Operand)
(8)	787	VAX\$CMPC5 - Compare Characters (5 Operand)
(9)	964	VAX\$SCANC - Scan Characters
(10)	1058	VAX\$SPANC - Span Characters
(11)	1152	VAX\$LOCC - Locate Character
(12)	1232	VAX\$SKPC - Skip Character
(13)	1313	VAX\$MATCHC - Match Characters
(14)	1451	VAX\$CRC - Calculate Cyclic Redundancy Check
(15)	1555	STRING_ACCVIO - Exception Dispatcher
(16)	1700	Packing Routines for String Instructions
(17)	1883	Packing Routines for MOVTC and MOVTC


```
0000 1 .NOSHOW CONDITIONALS
0000 3 .TITLE VAX$STRING VAX-11 Character String Instruction Emulation
0000 7 .IDENT /V04-001/
0000 8
0000 9
0000 10 :*****
0000 11 :*
0000 12 :* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 13 :* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 14 :* ALL RIGHTS RESERVED.
0000 15 :*
0000 16 :* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 17 :* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 18 :* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 19 :* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 20 :* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 21 :* TRANSFERRED.
0000 22 :*
0000 23 :* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 24 :* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 25 :* CORPORATION.
0000 26 :*
0000 27 :* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 28 :* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 29 :*
0000 30 :*
0000 31 :*****
0000 32 :
0000 33
0000 34 :++
0000 35 : Facility:
0000 36 :
0000 37 : VAX-11 Instruction Emulator
0000 38 :
0000 39 : Abstract:
0000 40 :
0000 41 : The routines in this module emulate the VAX-11 string instructions.
0000 42 : These procedures can be a part of an emulator package or can be
0000 43 : called directly after the input parameters have been loaded into
0000 44 : the architectural registers.
0000 45 :
0000 46 : The input parameters to these routines are the registers that
0000 47 : contain the intermediate instruction state.
0000 48 :
0000 49 : Environment:
0000 50 :
0000 51 : These routines run at any access mode, at any IPL, and are AST
0000 52 : reentrant.
0000 53 :
0000 54 : Author:
0000 55 :
0000 56 : Lawrence J. Kenah
0000 57 :
0000 58 : Creation Date:
0000 59 :
0000 60 : 16 August 1982
0000 61 :
```



```
0000 62 : Modified by:
0000 63 :
0000 64 : V04-001 LJK0044 Lawrence J. Kenah 6-Sep-1984
0000 65 : The backup code for MOVTC when moving in the forward direction
0000 66 : also needs to be changed (see LJK0039) based on the relative
0000 67 : sizes of the source and destination strings.
0000 68 :
0000 69 : V01-005 KDM0107 Kathleen D. Morse 21-Aug-1984
0000 70 : Fix bug in CMPC3. Return C clear if string length is 0.
0000 71 :
0000 72 : V01-004 LJK0039 Lawrence J. Kenah 20-Jul-1984
0000 73 : Modify MOVTC backup code to reflect differences in register
0000 74 : contents when traversing strings backwards. There are two
0000 75 : cases based on the relative sizes of source and destination.
0000 76 :
0000 77 : V01-003 LJK0026 Lawrence J. Kenah 19-Mar-1984
0000 78 : Final cleanup pass. Access violation handler is now called
0000 79 : STRING ACCVIO. Set PACK M ACCVIO bit in R1 before passing
0000 80 : control to VAX$REFLECT_FAULT.
0000 81 :
0000 82 : V01-002 LJK0011 Lawrence J. Kenah 8-Nov-1983
0000 83 : Fix three minor bugs in MOVTC and MOVTUC. Change exception
0000 84 : handling to reflect changed implementation.
0000 85 :
0000 86 : V01-001 Original Lawrence J. Kenah 16-Aug-1982
0000 87 :--
```



```
0000 89 .SUBTITLE Miscellaneous Notes
0000 90 :+
0000 91 : The following notes apply to most or all of the routines that appear in
0000 92 : this module. The comments appear here to avoid duplication in each routine.
0000 93 :
0000 94 : 1. The VAX Architecture Standard (DEC STD 032) is the ultimate authority on
0000 95 : the functional behavior of these routines. A summary of each instruction
0000 96 : that is emulated appears in the Functional Description section of each
0000 97 : routine header.
0000 98 :
0000 99 : 2. One design goal that affects the algorithms used is that these instructions
0000 100 : can incur exceptions such as access violations that will be reported to
0000 101 : users in such a way that the exception appears to have originated at the
0000 102 : site of the reserved instruction rather than within the emulator. This
0000 103 : constraint affects the algorithms available and dictates specific
0000 104 : implementation decisions.
0000 105 :
0000 106 : 3. Each routine header contains a picture of the register usage when it is
0000 107 : necessary to store the intermediate state of an instruction (routine) while
0000 108 : servicing an exception.
0000 109 :
0000 110 : The delta-PC field is used by the condition handler jacket to these
0000 111 : routines when it determines that an exception such as an access violation
0000 112 : occurred in response to an explicit use of one of the reserved
0000 113 : instructions. These routines can also be called directly with the input
0000 114 : parameters correctly placed in registers. The delta-PC field is not used in
0000 115 : this case.
0000 116 :
0000 117 : Note that the input parameters to any routine are a subset of the
0000 118 : intermediate state picture.
0000 119 :
0000 120 : Fields that are not used either as input parameters or to store
0000 121 : intermediate state are indicated thus, XXXXX.
0000 122 :
0000 123 : 4. In the Input Parameter list for each routine, certain register fields that
0000 124 : are not used may be explicitly listed for one reason or another. These
0000 125 : unused input parameters are described as IRRELEVANT.
0000 126 :
0000 127 : 5. In general, the final condition code settings are determined as the side
0000 128 : effect of one of the last instructions that executes before control is
0000 129 : passed back to the caller with an RSB. It is seldom necessary to explicitly
0000 130 : manipulate condition codes with a BIXPSW instruction or similar means.
0000 131 :
0000 132 : 6. There is only a small set of exceptions that are reflected to the user in an
0000 133 : altered fashion, with the exception PC changed from within the emulator to
0000 134 : the site of the original entry into these routines. The instructions that
0000 135 : generate these exceptions are all immediately preceded by a
0000 136 :
0000 137 : MARK_POINT yyyy_N
0000 138 :
0000 139 : where yyyy is the instruction name and N is a small integer. These names
0000 140 : map directly into instruction- and context-specific routines (located at
0000 141 : the end of this module) that put each instruction (routine) into a
0000 142 : consistent state before passing control to a more general exception handler
0000 143 : in a different module.
0000 144 :-
```



```
0000 146      .SUBTITLE      DECLARATIONS
0000 147
0000 148 ; Include files:
0000 149
0000 150      $PSLDEF          ; Define bit fields in PSL
0000 151
0000 152      .NOCROSS        ; No cross reference for these
0000 153      .ENABLE        SUPPRESSION ; No symbol table entries either
0000 154
0000 155      PACK_DEF        ; Stack usage for exception handling
0000 156
0000 157      .DISABLE        SUPPRESSION ; Turn on symbol table again
0000 158      .CROSS          ; Cross reference is OK now
0000 159
0000 160 ; Macro Definitions
0000 161
0000 162      .MACRO      INCLUDE      OPCODE , BOOT_FLAG
0000 163      .IF      NOT_DEFINED      BOOT_SWITCH
0000 164      OPCODE' _DEF
0000 165      INCLUDE_' OPCODE = 0
0000 166      .IF_FALSE
0000 167      .IF      IDENTICAL      <BOOT_FLAG> , BOOT
0000 168      OPCODE' _DEF
0000 169      INCLUDE_' OPCODE = 0
0000 170      .ENDC
0000 171      .ENDC
0000 172      .ENDM      _INCLUDE
0000 173
0000 174 ; External declarations
0000 175
0000 176      .DISABLE      GLOBAL
0000 177
0000 179      .EXTERNAL      VAX$REFLECT_FAULT
0000 181
0000 182 ; PSECT Declarations:
0000 183
0000 184      .DEFAULT      DISPLACEMENT , WORD
0000 185
00000000 186      .PSECT _VAX$CODE PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, LONG
0000 187
0000 188      BEGIN_MARK_POINT      ; Set up exception mark points
```

```
0000 190      .SUBTITLE      Conditional Assembly Parameters
0000 191      ;+
0000 192      ; Functional Description:
0000 193      ;
0000 194      ; It is possible to create a subset emulator, one that emulates
0000 195      ; specific reserved instructions. This capability is currently exploited
0000 196      ; to create a subset emulator for use by the bootstrap programs.
0000 197      ;
0000 198      ; An instruction is included in the full emulator by making an entry
0000 199      ; in the following table. If the optional second parameter is present
0000 200      ; and equal to BOOT, then that instruction is included in the subset
0000 201      ; emulator used by the bootstrap code.
0000 202      ;-
0000 203
0000 204      .NOCROSS          ; No cross reference for these
0000 205      .ENABLE          SUPPRESSION      ; No symbol table entries either
0000 206
0000 207      -INCLUDE          MOVTC
0000 208      -INCLUDE          MOVTC
0000 209      -INCLUDE          CMPC3 , BOOT
0000 210      -INCLUDE          CMPC5 , BOOT
0000 211      -INCLUDE          SCANC
0000 212      -INCLUDE          SPANC
0000 213      -INCLUDE          LOCC , BOOT
0000 214      -INCLUDE          SKPC
0000 215      -INCLUDE          MATCHC
0000 216      -INCLUDE          CRC
0000 217
0000 218      .DISABLE          SUPPRESSION      ; Turn on symbol table again
0000 219      .CROSS          ; Cross reference is OK now
0000 220
0000 221      .NOSHOW          CONDITIONALS
0000 222
```



```
0000 225 .SUBTITLE VAX$MOVTC - Move Translated Characters
0000 226 :+
0000 227 : Functional Description:
0000 228 :
0000 229 : The source string specified by the source length and source address
0000 230 : operands is translated and replaces the destination string specified by
0000 231 : the destination length and destination address operands. Translation is
0000 232 : accomplished by using each byte of the source string as an index into a
0000 233 : 256 byte table whose zeroth entry address is specified by the table
0000 234 : address operand. The byte selected replaces the byte of the destination
0000 235 : string. If the destination string is longer than the source string, the
0000 236 : highest addressed bytes of the destination string are replaced by the
0000 237 : fill operand. If the destination string is shorter than the source
0000 238 : string, the highest addressed bytes of the source string are not
0000 239 : translated and moved. The operation of the instruction is such that
0000 240 : overlap of the source and destination strings does not affect the
0000 241 : result. If the destination string overlaps the translation table, the
0000 242 : destination string is UNPREDICTABLE.
0000 243 :
0000 244 : Input Parameters:
0000 245 :
0000 246 : The following register fields contain the same information that
0000 247 : exists in the operands to the MOVTC instruction.
0000 248 :
0000 249 : R0<15:0> = srclen Length of source string
0000 250 : R1 = srcaddr Address of source string
0000 251 : R2<7:0> = fill Fill character
0000 252 : R3 = tbladdr Address of 256-byte table
0000 253 : R4<15:0> = dstlen Length of destination string
0000 254 : R5 = dstaddr Address of destination string
0000 255 :
0000 256 : In addition to the input parameters that correspond directly to
0000 257 : operands to the MOVTC instruction, there are other input parameters
0000 258 : to this routine. Note that the two inixxlen parameters are only
0000 259 : used when the MOVTC_V_FPD bit is set in the FLAGS byte.
0000 260 :
0000 261 : R2<15:8> = FLAGS Instruction-specific status
0000 262 :
0000 263 : The contents of the FLAGS byte must be zero (MBZ) on entry to this
0000 264 : routine from the outside world (through the emulator jacket or by
0000 265 : a JSB call). If the initial contents of FLAGS are not zero, the
0000 266 : actions of this routine are UNPREDICTABLE.
0000 267 :
0000 268 : There are two other input parameters whose contents depend on
0000 269 : the settings of the FLAGS byte.
0000 270 :
0000 271 : MOVTC_V_FPD bit in FLAGS is CLEAR
0000 272 :
0000 273 : R0<31:16> = IRRELEVANT
0000 274 : R4<31:16> = IRRELEVANT
0000 275 :
0000 276 : MOVTC_V_FPD bit in FLAGS is SET
0000 277 :
0000 278 : R0<31:16> = inisrclen Initial length of source string
0000 279 : R4<31:16> = inidstlen Initial length of destination string
0000 280 :
0000 281 : Intermediate State:
```


31	23	15	07	00	
initial srclen		srclen			: R0
srcaddr					: R1
delta-PC	XXXX	FLAGS	fill		: R2
tbladdr					: R3
initial dstlen		dstlen			: R4
dstaddr					: R5

Output Parameters:

Source string longer than destination string

R0 = Number of bytes remaining in the source string
R1 = Address of one byte beyond last byte in source string that was translated (the first untranslated byte)
R2 = 0
R3 = tbladdr Address of 256-byte table
R4 = 0 (Number of bytes remaining in the destination string)
R5 = Address of one byte beyond end of destination string

Source string same size as or smaller than destination string

R0 = 0 (Number of bytes remaining in the source string)
R1 = Address of one byte beyond end of source string
R2 = 0
R3 = tbladdr Address of 256-byte table
R4 = 0 (Number of bytes remaining in the destination string)
R5 = Address of one byte beyond end of destination string

Condition Codes:

N <- srclen LSS dstlen
Z <- srclen EQL dstlen
V <- 0
C <- srclen LSSU dstlen

Side Effects:

This routine uses up to four longwords of stack space.

.ENABLE LOCAL_BLOCK

VAX\$MOVTC::

54 DD 0000 334 PUSHL R4 ; Store dstlen on stack
50 DD 0002 335 PUSHL R0 ; Store srclen on stack
0004 336
0004 337 ASSUME MOVTC_B_FLAGS EQ 9 ; Insure that FLAGS are in R2<15:8>
0004 338

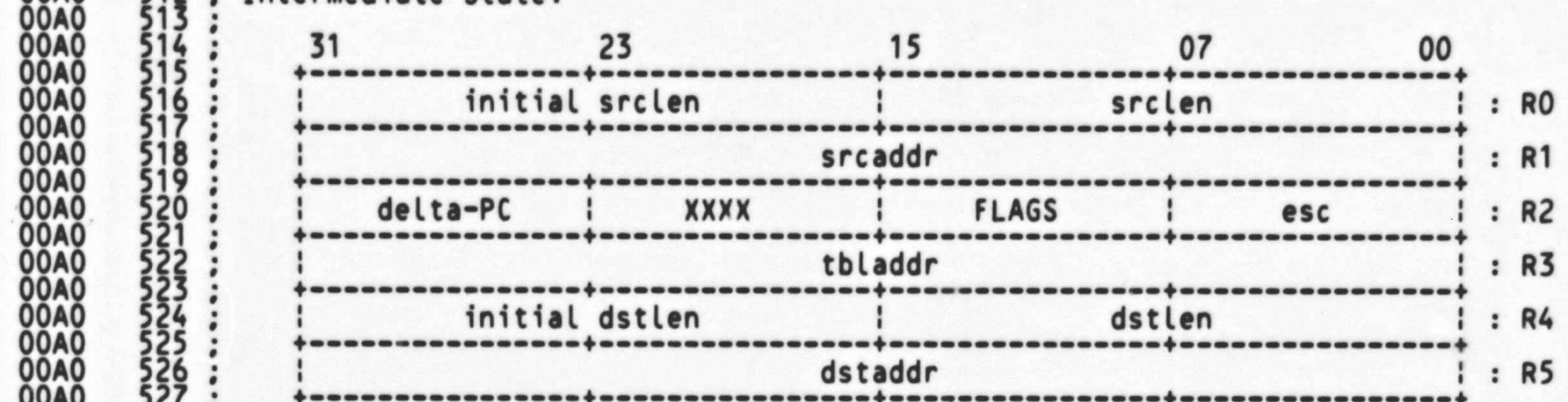

```
09 52 08 E0 0004 339 BBS #<MOVTC V FPD+8>,R2,5$ ; Branch if instruction was interrupted
02 AE 6E B0 0008 340 MOVW (SP),2(SP) ; Set the initial srclen on stack
06 AE 04 AE B0 000C 341 MOVW 4(SP),6(SP) ; Set the initial dstlen on stack
5A DD 0011 342 5$: PUSHL R10 ; Save R10 so it can hold handler
0013 343 ESTABLISH_HANDLER -
0013 344 STRING_ACCVIO ; Store address of condition handler
54 54 3C 0018 345 MOVZWL R4,R4 ; Clear unused bits of dstlen
41 13 001B 346 BEQL 40$ ; All done if zero
50 50 3C 001D 347 MOVZWL R0,R0 ; Clear unused bits of srclen
21 13 0020 348 BEQL 20$ ; Add fill character to destination
55 51 D1 0022 349 CMPL R1,R5 ; Check relative position of strings
3C 1F 0025 350 BLSSU MOVE_BACKWARD ; Perform move from end of strings
0027 351
0027 352 ; This code executes if the source string is at a LARGER virtual address
0027 353 ; than the destination string. The movement takes place from the front
0027 354 ; (small address end) of each string to the back (high address end).
0027 355
0027 356 MOVE_FORWARD:
54 52 DD 0027 357 PUSHL R2 ; Allow R2 (fill) to be used as scratch
50 50 C2 0029 358 SUBL R0,R4 ; Get difference between strings
04 1E 002C 359 BGEQU 10$ ; Branch if fill work to do eventually
50 0C AE 3C 002E 360 MOVZWL 12(SP),R0 ; Use dstlen (saved R4) as srclen (R0)
0032 361
0032 362 MARK_POINT MOVTC_1
52 81 9A 0032 363 10$: MOVZBL (R1)+,R2 ; Get next character from source
0035 364 MARK_POINT MOVTC_2
85 6342 90 0035 365 MOVB (R3)[R2],(R5)+ ; Move translated character
F6 50 F5 0039 366 SOBGTR R0,10$ ; Source all done?
003C 367
52 8E D0 003C 368 MOVL (SP)+,R2 ; Retrieve fill character from stack
54 D5 003F 369 TSTL R4 ; Do we need to fill anything?
56 15 0041 370 BLEQ 80$ ; Skip to exit code if no fill work
0043 371
0043 372 MARK_POINT MOVTC_3
85 52 90 0043 373 20$: MOVB R2,(R5)+ ; Fill next character
FA 54 F5 0046 374 SOBGTR R4,20$ ; Destination all done?
0049 375
0049 376 ; This is the common exit path. R2 is cleared to conform to its output
0049 377 ; setting. The condition codes are determined by the original lengths
0049 378 ; of the source and destination strings that were saved on the stack.
0049 379
0049 380 30$: CLRL R2 ; R2 is zero on return
04 AE 6E 6E 5A 8E D0 004B 381 MOVL (SP)+,R10 ; Restore saved R10
04 AE 04 AE F0 8F 78 004E 382 ASHL #-16,(SP),(SP) ; Get initial srclen
F0 8F 78 0053 383 ASHL #-16,4(SP),4(SP) ; Get initial dstlen
8E 8E D1 005A 384 CMPL (SP)+,(SP)+ ; Set condition codes
05 005D 385 RSB
005E 386
005E 387 ; The following instruction is the exit path when the destination string
005E 388 ; has zero length on input.
005E 389
50 50 3C 005E 390 40$: MOVZWL R0,R0 ; Clear unused bits of srclen
E6 11 0061 391 BRB 30$ ; Exit through common code
0063 392
0063 393 ; This code executes if the source string is at a SMALLER virtual address
0063 394 ; than the destination string. The movement takes place from the back
0063 395 ; (high address end) of each string to the front (low address end).
```



```
0063 396
0063 397 MOVE_BACKWARD:
55 54 C0 0063 398 ADDL R4,R5 ; Point R5 one byte beyond destination
54 50 C2 0066 399 SUBL R0,R4 ; Get amount of fill work to do
06 06 1A 0069 400 BGTRU 50$ ; Branch to fill loop if work to do
50 08 AE 3C 006B 401 MOVZWL 8(SP),R0 ; Use dstlen (saved R4) as srclen (R0)
06 06 11 006F 402 BRB 60$ ; Skip loop that does fill characters
0071 403
0071 404 MARK_POINT MOVTC_4
75 52 90 0071 405 50$: MOVBL R2,-(R5) ; Load fill characters from the back
FA 54 F5 0074 406 SOBGTR R4,50$ ; Continue until excess all done
0077 407
51 50 C0 0077 408 60$: ADDL R0,R1 ; Point R1 to 'modified end' of source
007A 409
007A 410 ; Move transtaled characters from the high-address end toward the low-address
007A 411 ; end. Note that the fill character is no longer needed so that R2 is
007A 412 ; available as a scratch register.
007A 413
007A 414 MARK_POINT MOVTC_5
52 71 9A 007A 415 70$: MOVZBL -(R1),R2 ; Get next character
007D 416 MARK_POINT MOVTC_6
75 6342 90 007D 417 MOVBL (R3)[R2],-(R5) ; Move translated character
F6 50 F5 0081 418 SOBGTR R0,70$ ; Continue until source is exhausted
0084 419
0084 420 ; At this point, R1 points to the first character in the source string and R5
0084 421 ; points to the first character in the destination string. This is the result
0084 422 ; of operating on the strings from back to front (high-address end to
0084 423 ; low-address end). These registers must be modified to point to the ends of
0084 424 ; their respective strings. This is accomplished by using the saved original
0084 425 ; lengths of the two strings. Note that at this stage of the routine, R2 is
0084 426 ; no longer needed and so can be used as a scratch register.
0084 427
52 06 AE 3C 0084 428 MOVZWL 6(SP),R2 ; Get original source length
51 52 C0 0088 429 ADDL R2,R1 ; Point R1 to end of source string
52 0A AE 3C 008B 430 MOVZWL 10(SP),R2 ; Get original destination length
55 52 C0 008F 431 ADDL R2,R5 ; Point R5 to end of destination string
0092 432
0092 433 ; If R1 is negative, this indicates that the source string is smaller than the
0092 434 ; destination. R1 must be readjusted to point to the first byte that was not
0092 435 ; translated. R0, which contains zero, must be loaded with the number of bytes
0092 436 ; that were not translated (the negative of the contents of R4).
0092 437
0092 438 TSTL R4 ; Any more work to do?
51 54 D5 0092 438 TSTL R4 ; Any more work to do?
B3 13 0094 439 BEQL 30$ ; Exit through common code
54 54 C0 0096 440 ADDL R4,R1 ; Back up R1 (R4 is negative)
0099 441
0099 442 ; The exit code for MOVE_FORWARD also comes here if the source is longer than
0099 443 ; (or equal to) the destination. Note that in the case of R4 containing zero,
0099 444 ; some extra work that accomplishes nothing must be done. This extra work in
0099 445 ; the case of equal strings avoids two extra instructions in all cases.
0099 446
50 54 CE 0099 447 80$: MNEGL R4,R0 ; Remaining source length to R0
54 54 D4 009C 448 CLRL R4 ; R4 is always zero on exit
A9 11 009E 449 BRB 30$ ; Exit through common code
00A0 450
00A0 451 .DISABLE LOCAL_BLOCK
```


00A0 455 .SUBTITLE VAX\$MOVTUC - Move Translated Until Character
00A0 456 :+
00A0 457 : Functional Description:
00A0 458 :
00A0 459 : The source string specified by the source length and source address
00A0 460 : operands is translated and replaces the destination string specified by
00A0 461 : the destination length and destination address operands. Translation is
00A0 462 : accomplished by using each byte of the source string as index into a 256
00A0 463 : byte table whose zeroth entry address is specified by the table address
00A0 464 : operand. The byte selected replaces the byte of the destination string.
00A0 465 : Translation continues until a translated byte is equal to the escape
00A0 466 : byte or until the source string or destination string is exhausted. If
00A0 467 : translation is terminated because of escape the condition code V-bit is
00A0 468 : set; otherwise it is cleared. If the destination string overlaps the
00A0 469 : table, the destination string and registers R0 through R5 are
00A0 470 : UNPREDICTABLE. If the source and destination strings overlap and their
00A0 471 : addresses are not identical, the destination string and registers R0
00A0 472 : through R5 are UNPREDICTABLE. If the source and destination string
00A0 473 : addresses are identical, the translation is performed correctly.
00A0 474 :
00A0 475 : Input Parameters:
00A0 476 :
00A0 477 : The following register fields contain the same information that
00A0 478 : exists in the operands to the MOVTUC instruction.
00A0 479 :
00A0 480 : R0<15:0> = srclen Length of source string
00A0 481 : R1 = srcaddr Address of source string
00A0 482 : R2<7:0> = fill Escape character
00A0 483 : R3 = tbladdr Address of 256-byte table
00A0 484 : R4<15:0> = dstlen Length of destination string
00A0 485 : R5 = dstaddr Address of destination string
00A0 486 :
00A0 487 : In addition to the input parameters that correspond directly to
00A0 488 : operands to the MOVTUC instruction, there are other input parameters
00A0 489 : to this routine. Note that the two inixxlen parameters are only
00A0 490 : used when the MOVTUC_V_FPD bit is set in the FLAGS byte.
00A0 491 :
00A0 492 : R2<15:8> = FLAGS Instruction-specific status
00A0 493 :
00A0 494 : The contents of the FLAGS byte must be zero (MBZ) on entry to this
00A0 495 : routine from the outside world (through the emulator jacket or by
00A0 496 : a JSB call). If the initial contents of FLAGS are not zero, the
00A0 497 : actions of this routine are UNPREDICTABLE.
00A0 498 :
00A0 499 : There are two other input parameters whose contents depend on
00A0 500 : the settings of the FLAGS byte.
00A0 501 :
00A0 502 : MOVTUC_V_FPD bit in FLAGS is CLEAR
00A0 503 :
00A0 504 : R0<31:16> = IRRELEVANT
00A0 505 : R4<31:16> = IRRELEVANT
00A0 506 :
00A0 507 : MOVTUC_V_FPD bit in FLAGS is SET
00A0 508 :
00A0 509 : R0<31:16> = inisrclen Initial length of source string
00A0 510 : R4<31:16> = inidstlen Initial length of destination string
00A0 511 :

00A0 512 : Intermediate State:

00A0 528 :
00A0 529 : Output Parameters:

00A0 530 :
00A0 531 : The final state of this instruction (routine) can exist in one of
00A0 532 : three forms, depending on the relative lengths of the source and
00A0 533 : destination strings and whether a translated character matched the
00A0 534 : escape character.

00A0 535 :
00A0 536 : 1. Some byte matched escape character

00A0 537 : R0 = Number of bytes remaining in the source string (including
00A0 538 : the byte that caused the escape)
00A0 539 : R1 = Address of the byte that caused the escape
00A0 540 : R2 = 0
00A0 541 : R3 = tbladdr Address of 256-byte table
00A0 542 : R4 = Number of bytes remaining in the destination string
00A0 543 : R5 = Address of byte that would have received the translated byte

00A0 544 :
00A0 545 : 2. Destination string exhausted

00A0 546 : R0 = Number of bytes remaining in the source string
00A0 547 : R1 = Address of the byte that resulted in exhaustion
00A0 548 : R2 = 0
00A0 549 : R3 = tbladdr Address of 256-byte table
00A0 550 : R4 = 0 (Number of bytes remaining in the destination string)
00A0 551 : R5 = Address of one byte beyond end of destination string

00A0 552 :
00A0 553 : 3. Source string exhausted

00A0 554 : R0 = 0 (Number of bytes remaining in the source string)
00A0 555 : R5 = Address of one byte beyond end of source string
00A0 556 : R2 = 0
00A0 557 : R3 = tbladdr Address of 256-byte table
00A0 558 : R4 = Number of bytes remaining in the destination string
00A0 559 : R5 = Address of byte that would have received the translated byte

00A0 560 :
00A0 561 : Condition Codes:

00A0 562 : N <- srclen LSS dstlen
00A0 563 : Z <- srclen EQL dstlen
00A0 564 : V <- set if terminated by escape
00A0 565 :
00A0 566 :
00A0 567 :
00A0 568 :


```
00A0 569 : C <- srclen LSSU dstlen
00A0 570 :
00A0 571 : Side Effects:
00A0 572 :
00A0 573 : This routine uses five longwords of stack.
00A0 574 :-
00A0 575 :
00A0 576 : .ENABLE LOCAL_BLOCK
00A0 577 :
00A0 578 VAX$MOVTUC::
54 DD 00A0 579 PUSHL R4 ; Store dstlen on stack
50 DD 00A2 580 PUSHL R0 ; Store srclen on stack
00A4 581
00A4 582 ASSUME MOVTUC_B_FLAGS EQ 9 ; Insure that FLAGS are in R2<15:8>
00A4 583
09 52 08 E0 00A4 584 BBS #<MOVTUC_V_FPD+8>,R2,5$ ; Branch if instruction was interrupted
02 AE 6E B0 00A8 585 MOVW (SP),2(SP) ; Set the initial srclen on stack
06 AE 04 AE B0 00AC 586 MOVW 4(SP),6(SP) ; Set the initial dstlen on stack
54 54 3C 00B1 587 5$: MOVZWL R4,R4 ; Clear unused bits of dstlen
4F 13 00B4 588 BEQL 50$ ; Almost done if zero length
50 50 3C 00B6 589 MOVZWL R0,R0 ; Clear unused bits of srclen
38 13 00B9 590 BEQL 40$ ; Done if zero length
5A DD 00BB 591 PUSHL R10 ; Save R10 so it can hold handler
00BD 592 ESTABLISH_HANDLER -
00BD 593 STRING_ACCVIO ; Store address of condition handler
57 DD 00C2 594 PUSHL R7 ; We need some scratch registers
56 DD 00C4 595 PUSHL R6 ;
00C6 596
00C6 597 ; Note that all code must now exit through a code path that restores R6
00C6 598 ; R7, and R10 to insure that the stack is correctly aligned and that these
00C6 599 ; register contents are preserved across execution of this routine.
00C6 600
00C6 601 ; The following initialization routine is designed to make the main loop
00C6 602 ; execute faster. It performs three actions.
00C6 603 :
00C6 604 : R7 <- Smaller of R0 and R4 (srclen and dstlen)
00C6 605 :
00C6 606 : Larger of R0 and R4 is replaced by the difference between R0 and R4.
00C6 607 :
00C6 608 : Smaller of R0 and R4 is replaced by zero.
00C6 609 :
00C6 610 : This initializes R0 and R4 to their final states if either the source
00C6 611 : string or the destination string is exhausted. In the event that the loop
00C6 612 : is terminated through the escape path, these two registers are readjusted
00C6 613 : to contain the proper values as if they had each been advanced one byte
00C6 614 : for each trip through the loop.
00C6 615
54 50 C2 00C6 616 SUBL R0,R4 ; Replace R4 with (R4-R0)
07 1F 00C9 617 BLSSU 10$ ; Branch if srclen GTRU dstlen
00CB 618
00CB 619 ; Code path for srclen (R0) LEQU dstlen (R4). R4 is already correctly loaded.
00CB 620
57 50 D0 00CB 621 MOVL R0,R7 ; Load R7 with smaller (R0)
50 D4 00CE 622 CLRL R0 ; Load smaller (R0) with zero
09 11 00D0 623 BRB 20$ ; Merge with common code at top of loop
00D2 624
00D2 625 ; Code path for srclen (R0) GTRU dstlen (R4).
```



```

57 10 AE 3C 00D2 626
50 54 CE 00D6 627 10$: MOVZWL 16(SP),R7 ; Load R7 with smaller (use saved R4)
54 54 D4 00D9 628 MNEGL R4,R0 ; Load larger (R0) with ABS(R4-R0)
00DB 629 CLRL R4 ; Load smaller (R4) with zero
00DB 630
00DB 631 ; The following is the main loop in this routine.
00DB 632
00DB 633
56 81 9A 00DB 634 20$: MARK POINT MOVTUC_1 ; Get next character from source string
00DE 635 MOVZBL (R1)+,R6
56 6346 9A 00DE 636 MARK POINT MOVTUC_2 ; Convert to translated character
56 52 91 00E2 637 MOVZBL (R3)[R6],R6 ; Does it match escape character?
23 13 00E5 638 CMPB R2,R6 ; Exit loop if yes
00E7 639 BEQL ESCAPE
85 56 90 00E7 640 MARK_POINT MOVTUC_3 ; Move translated character to
00EA 641 MOVB R6,(R5)+ ; destination string
EE 57 F5 00EA 642 SOBGTR R7,20$ ; Shorter string exhausted?
00ED 643
00ED 644 ; The following exit path is taken when the shorter of the source string and
00ED 645 ; the destination string is exhausted
00ED 646
56 8E 7D 00ED 647 30$: MOVQ (SP)+,R6 ; Restore contents of scratch register
5A 8E D0 00F0 648 MOVL (SP)+,R10 ; Restore saved R10
52 D4 00F3 649 40$: CLRL R2 ; R2 must be zero on output
04 AE 6E 04 AE F0 8F 78 00F5 650 ASHL #-16,(SP),(SP) ; Get initial srclen
8E 8E D1 00FA 651 ASHL #-16,4(SP),4(SP) ; Get initial dstlen
05 0101 652 CMPL (SP)+,(SP)+ ; Set condition codes (V-bit always 0)
0104 653 RSB ; Return
0105 654
0105 655 ; This code executes if the destination string has zero length. The source
0105 656 ; length is set to a known state so that the common exit path can be taken.
0105 657
50 50 3C 0105 658 50$: MOVZWL R0,R0 ; Clear unused bits of srclen
E9 11 0108 659 BRB 40$ ; Exit through common code
010A 660
010A 661 ; This code executes if the escape character matches the entry in the
010A 662 ; 256-byte table indexed by the character in the source string. Registers
010A 663 ; R0 and R4 must be adjusted to indicate that neither string was exhausted.
010A 664 ; The last step taken before return sets the V-bit.
010A 665
010A 666 ESCAPE:
51 D7 010A 667 DECL R1 ; Reset R1 to correct byte in source
52 D4 010C 668 CLRL R2 ; R2 must be zero on output
50 57 C0 010E 669 ADDL R7,R0 ; Adjust saved srclen
54 57 C0 0111 670 ADDL R7,R4 ; Adjust saved dstlen
56 8E 7D 0114 671 MOVQ (SP)+,R6 ; Restore contents of scratch registers
5A 8E D0 0117 672 MOVL (SP)+,R10 ; Restore saved R10
04 AE 6E 04 AE F0 8F 78 011A 673 ASHL #-16,(SP),(SP) ; Get initial srclen
8E 8E D1 011F 674 ASHL #-16,4(SP),4(SP) ; Get initial dstlen
02 B8 0126 675 CMPL (SP)+,(SP)+ ; Set condition codes (V-bit always 0)
05 0129 676 BISPSW #PSL$M_V ; Set V-bit to indicate ESCAPE
012B 677 RSB ; Return
012C 678
012C 679 .DISABLE LOCAL_BLOCK
```



```
012C 683 .SUBTITLE VAX$CMPC3 - Compare Characters (3 Operand)
012C 684 :+
012C 685 : Functional Description:
012C 686 :
012C 687 : The bytes of string 1 specified by the length and address 1 operands are
012C 688 : compared with the bytes of string 2 specified by the length and address
012C 689 : 2 operands. Comparison proceeds until inequality is detected or all the
012C 690 : bytes of the strings have been examined. Condition codes are affected
012C 691 : by the result of the last byte comparison. Two zero length strings
012C 692 : compare equal (i.e. Z is set and N, V, and C are cleared).
012C 693 :
012C 694 : Input Parameters:
012C 695 :
012C 696 : R0<15:0> = len Length of character strings
012C 697 : R1 = src1addr Address of first character string (called S1)
012C 698 : R3 = src2addr Address of second character string (called S2)
012C 699 :
012C 700 : Intermediate State:
012C 701 :
012C 702 : 31 23 15 07 00
012C 703 : +-----+-----+-----+-----+
012C 704 : | delta-PC | XXXX | | len | : R0
012C 705 : +-----+-----+-----+-----+
012C 706 : | | | src1addr | : R1
012C 707 : +-----+-----+-----+-----+
012C 708 : | | | XXXXX | : R2
012C 709 : +-----+-----+-----+-----+
012C 710 : | | | src2addr | : R3
012C 711 : +-----+-----+-----+-----+
012C 712 :
012C 713 : Output Parameters:
012C 714 :
012C 715 : Strings are IDENTICAL
012C 716 :
012C 717 : R0 = 0
012C 718 : R1 = Address of one byte beyond end of S1
012C 719 : R2 = 0 (same as R0)
012C 720 : R3 = Address of one byte beyond end of S2
012C 721 :
012C 722 : Strings DO NOT MATCH
012C 723 :
012C 724 : R0 = Number of bytes left in strings (including first byte
012C 725 : that did not match)
012C 726 : R1 = Address of nonmatching byte in S1
012C 727 : R2 = R0
012C 728 : R3 = Address of nonmatching byte in S2
012C 729 :
012C 730 : Condition Codes:
012C 731 :
012C 732 : In general, the condition codes reflect whether or not the strings
012C 733 : are considered the same or different. In the case of different
012C 734 : strings, the condition codes reflect the result of the comparison
012C 735 : that indicated that the strings are not equal.
012C 736 :
012C 737 : Strings are IDENTICAL
012C 738 :
012C 739 : N <- 0
```



```
012C 740 : Z <- 1 ; (byte in S1) EQL (byte in S2)
012C 741 : V <- 0
012C 742 : C <- 0
012C 743 :
012C 744 : Strings DO NOT MATCH
012C 745 :
012C 746 : N <- (byte in S1) LSS (byte in S2)
012C 747 : Z <- 0 ; (byte in S1) NEQ (byte in S2)
012C 748 : V <- 0
012C 749 : C <- (byte in S1) LSSU (byte in S2)
012C 750 :
012C 751 : where "byte in S1" or "byte in S2" may indicate the fill character
012C 752 :
012C 753 : Side Effects:
012C 754 :
012C 755 : This routine uses one longword of stack.
012C 756 :-
012C 757
012C 758 VAX$CMPC3::
50 50 3C 012C 759 MOVZWL R0,R0 ; Clear unused bits & check for zero
12 13 012F 760 BEQL 20$ ; Simply return if zero length string
5A DD 0131 761
0133 762 PUSHL R10 ; Save R10 so it can hold handler
0133 763 ESTABLISH HANDLER -
0138 764 STRING_ACCVIO ; Store address of condition handler
0138 765
0138 766 MARK_POINT CMPC3_1
81 83 91 0138 767 10$: CMPB (R3)+,(R1)+ ; Character match?
0B 12 013B 768 BNEQ 30$ ; Exit loop if different
F8 50 F5 013D 769 SOBGTR R0,10$
0140 770
0140 771 ; Exit path for strings IDENTICAL (R0 = 0, either on input or after loop)
0140 772
5A 8E D0 0140 773 MOVL (SP)+,R10 ; Restore saved R10
52 D4 0143 774 20$: CLRL R2 ; Set R2 for output value of 0
50 D5 0145 775 TSTL R0 ; Set condition codes
05 0147 776 RSB ; Return point for IDENTICAL strings
0148 777
0148 778 ; Exit path when strings DO NOT MATCH
0148 779
5A 8E D0 0148 780 30$: MOVL (SP)+,R10 ; Restore saved R10
52 50 D0 014B 781 MOVL R0,R2 ; R0 and R2 are the same on exit
73 71 91 014E 782 CMPB -(R1),-(R3) ; Reset R1 and R3 and set condition codes
05 0151 783 RSB ; Return point when strings DO NOT MATCH
```



```
0152 787 .SUBTITLE VAX$CMPC5 - Compare Characters (5 Operand)
0152 788 :+
0152 789 : Functional Description:
0152 790 :
0152 791 : The bytes of the string 1 specified by the length 1 and address 1
0152 792 : operands are compared with the bytes of the string 2 specified by the
0152 793 : length 2 and address 2 operands. If one string is longer than the
0152 794 : other, the shorter string is conceptually extended to the length of the
0152 795 : longer by appending (at higher addresses) bytes equal to the fill
0152 796 : operand. Comparison proceeds until inequality is detected or all the
0152 797 : bytes of the strings have been examined. Condition codes are affected
0152 798 : by the result of the last byte comparison. Two zero length strings
0152 799 : compare equal (i.e. Z is set and N, V, and C are cleared).
0152 800 :
0152 801 : Input Parameters:
0152 802 :
0152 803 : R0<15:0> = len Length of first character string (called S1)
0152 804 : R0<23:16> = fill Fill character that is used when strings have
0152 805 : different lengths
0152 806 : R1 = addr Address of first character string
0152 807 : R2<15:0> = len Length of second character string (called S2)
0152 808 : R3 = addr Address of second character string
0152 809 :
0152 810 : Intermediate State:
0152 811 :
0152 812 : 31 23 15 07 00
0152 813 : +-----+-----+-----+-----+
0152 814 : | delta-PC | fill | src1len | : R0
0152 815 : +-----+-----+-----+-----+
0152 816 : | | | src1addr | : R1
0152 817 : +-----+-----+-----+-----+
0152 818 : | XXXXX | | src2len | : R2
0152 819 : +-----+-----+-----+-----+
0152 820 : | | | src2addr | : R3
0152 821 : +-----+-----+-----+-----+
0152 822 :
0152 823 : Output Parameters:
0152 824 :
0152 825 : Strings are IDENTICAL
0152 826 :
0152 827 : R0 = 0
0152 828 : R1 = Address of one byte beyond end of S1
0152 829 : R2 = 0 (same as R0)
0152 830 : R1 = Address of one byte beyond end of S2
0152 831 :
0152 832 : Strings DO NOT MATCH
0152 833 :
0152 834 : R0 = Number of bytes remaining in S1 when mismatch detected
0152 835 : (or zero if S1 exhausted before mismatch detected)
0152 836 : R1 = Address of nonmatching byte in S1
0152 837 : R2 = Number of bytes remaining in S2 when mismatch detected
0152 838 : (or zero if S2 exhausted before mismatch detected)
0152 839 : R3 = Address of nonmatching byte in S2
0152 840 :
0152 841 : Condition Codes:
0152 842 :
0152 843 : In general, the condition codes reflect whether or not the strings
```



```
0152 844 : are considered the same or different. In the case of different
0152 845 : strings, the condition codes reflect the result of the comparison
0152 846 : that indicated that the strings are not equal.
0152 847 :
0152 848 : Strings are IDENTICAL
0152 849 :
0152 850 : N <- 0
0152 851 : Z <- 1 ; (byte in S1) EQL (byte in S2)
0152 852 : V <- 0
0152 853 : C <- 0
0152 854 :
0152 855 : Strings DO NOT MATCH
0152 856 :
0152 857 : N <- (byte in S1) LSS (byte in S2)
0152 858 : Z <- 0 ; (byte in S1) NEQ (byte in S2)
0152 859 : V <- 0
0152 860 : C <- (byte in S1) LSSU (byte in S2)
0152 861 :
0152 862 : where "byte in S1" or "byte in S2" may indicate the fill character
0152 863 :
0152 864 : Side Effects:
0152 865 :
0152 866 : This routine uses two longwords of stack.
0152 867 :-
0152 868 :
0152 869 : .ENABLE LOCAL_BLOCK
0152 870 :
0152 871 VAX$CMPC5::
5A DD 0152 872 PUSHL R10 ; Save R10 so it can hold handler
0154 873 ESTABLISH_HANDLER -
0154 874 STRING_ACCVIO ; Store address of condition handler
54 50 F0 54 DD 0159 875 PUSHL R4 ; Save register
50 50 3C 015B 876 ASHL #16,R0,R4 ; Get escape character
52 52 3C 0160 877 MOVZWL R0,R0 ; Clear unused bits & is S1 length zero?
14 13 0163 878 BEQL 50$ ; Branch if yes
0165 879 MOVZWL R2,R2 ; Clear unused bits & is S2 length zero?
0168 880 BEQL 30$
016A 881
016A 882 ; Main loop. The following loop executes when both strings have characters
016A 883 ; remaining and inequality has not yet been detected.
016A 884
016A 885 ; THE FOLLOWING LOOP IS A TARGET FOR FURTHER OPTIMIZATION IN THAT THE
016A 886 ; LOOP SHOULD NOT REQUIRE TWO SOBGR INSTRUCTIONS. NOTE, THOUGH, THAT
016A 887 ; THE CURRENT UNOPTIMIZED LOOP IS EASIER TO BACK UP.
016A 888
016A 889 MARK_POINT CMPC5_1
83 81 91 016A 890 10$: CMPB (R1)+,(R3)+ ; Characters match?
09 50 F5 016D 891 BNEQ 80$ ; Exit loop if bytes different
016F 892 SOBGR R0,20$ ; Check for S1 exhausted
0172 893
0172 894 ; The next test determines whether S2 is also exhausted.
0172 895
52 D7 0172 896 DECL R2 ; Put R2 in step with R0
1C 12 0174 897 BNEQ 60$ ; Branch if bytes remaining in S2
0176 898
0176 899 ; This is the exit path for identical strings. If we get here, then both
0176 900 ; R0 and R2 are zero. The condition codes are correctly set (by the ASHL
```



```
0176 901 ; instruction) so the registers are restored with a POPR to avoid changing
0176 902 ; the condition codes.
0176 903
0176 904 IDENTICAL:
0410 8F BA 0176 905 POPR #^M<R4,R10> ; Restore saved registers
05 017A 906 RSB ; Exit indicating IDENTICAL strings
017B 907
EC 52 F5 017B 908 20$: SOBGTR R2,10$ ; Check for S2 exhausted
017E 909
017E 910 ; The following loop is entered when all of S2 has been processed but
017E 911 ; there are characters remaining in S1. In other words,
017E 912 :
017E 913 : R0 GTRU 0
017E 914 : R2 EQL 0
017E 915 :
017E 916 ; The remaining characters in S1 are compared to the fill character.
017E 917
017E 918 MARK_POINT CMPC5_2
54 81 91 017E 919 30$: CMPB (R1)+,R4 ; Characters match?
05 12 0181 920 BNEQ 40$ ; Exit loop if no match
F8 50 F5 0183 921 SOBGTR R0,30$ ; Any more bytes in S1?
0186 922
EE 11 0186 923 BRB IDENTICAL ; Exit indicating IDENTICAL strings
0188 924
54 71 91 0188 925 40$: CMPB -(R1),R4 ; Reset R1 and set condition codes
17 11 018B 926 BRB NO_MATCH ; Exit indicating strings DO NOT MATCH
018D 927
018D 928 ; The following code executes if S1 has zero length on input. If S2 also
018D 929 ; has zero length, the routine simply returns, indicating equal strings.
018D 930
52 52 3C 018D 931 50$: MOVZWL R2,R2 ; Clear unused bits. Is S2 len also zero?
E4 13 0190 932 BEQL IDENTICAL ; Exit indicating IDENTICAL strings
0192 933
0192 934 ; The following loop is entered when all of S1 has been processed but
0192 935 ; there are characters remaining in S2. In other words,
0192 936 :
0192 937 : R0 EQL 0
0192 938 : R2 GTRU 0
0192 939 :
0192 940 ; The remaining characters in S2 are compared to the fill character.
0192 941
0192 942 MARK_POINT CMPC5_3
83 54 91 0192 943 60$: CMPB R4,(R3)+ ; Characters match?
05 12 0195 944 BNEQ 70$ ; Exit loop if no match
F8 52 F5 0197 945 SOBGTR R2,60$ ; Any more bytes in S2?
019A 946
DA 11 019A 947 BRB IDENTICAL ; Exit indicating IDENTICAL strings
019C 948
73 54 91 019C 949 70$: CMPB R4,-(R3) ; Reset R3 and set condition codes
03 11 019F 950 BRB NO_MATCH ; Exit indicating strings DO NOT MATCH
01A1 951
01A1 952 ; The following exit path is taken if both strings have characters
01A1 953 ; remaining and a character pair that did not match was detected.
01A1 954
73 71 91 01A1 955 80$: CMPB -(R1),-(R3) ; Reset R1 and R3 and set condition codes
01A4 956 NO_MATCH: ; Restore R4 and R10
0410 8F BA 01A4 957 POPR #^M<R4,R10> ; without changing condition codes
```


VAX\$STRING
V04-001

H 3
VAX-11 Character String Instruction Emul 16-SEP-1984 01:30:09 VAX/VMS Macro V04-00 Page 19
VAX\$CMPC5 - Compare Characters (5 Operan 7-SEP-1984 17:13:25 [EMULAT.SRC]VAXSTRING.MAR;2 (8)

05	01A8	958	RSB			
	01A9	959				
	01A9	960	.DISABLE	LOCAL_BLOCK		; Exit indicating strings DO NOT MATCH

01A9 964 .SUBTITLE VAX\$SCANC - Scan Characters
01A9 965 :+
01A9 966 : Functional Description:
01A9 967 :
01A9 968 : The bytes of the string specified by the length and address operands are
01A9 969 : successively used to index into a 256 byte table whose zeroth entry
01A9 970 : address is specified by the table address operand. The byte selected
01A9 971 : from the table is ANDed with the mask operand. The operation continues
01A9 972 : until the result of the AND is non-zero or all the bytes of the string
01A9 973 : have been exhausted. If a non-zero AND result is detected, the
01A9 974 : condition code Z-bit is cleared; otherwise, the Z-bit is set.
01A9 975 :
01A9 976 : Input Parameters:
01A9 977 :
01A9 978 : R0<15:0> = len Length of character string
01A9 979 : R1 = addr Address of character string
01A9 980 : R2<7:0> = mask Mask that is ANDed with successive characters
01A9 981 : R3 = tbladdr Address of 256-byte table
01A9 982 :
01A9 983 : Intermediate State:
01A9 984 :
01A9 985 : 31 23 15 07 00
01A9 986 : +-----+-----+-----+-----+
01A9 987 : | delta-PC | XXXX | | len | : R0
01A9 988 : +-----+-----+-----+-----+
01A9 989 : | | | addr | : R1
01A9 990 : +-----+-----+-----+-----+
01A9 991 : | XXXXX | | XXXX | mask | : R2
01A9 992 : +-----+-----+-----+-----+
01A9 993 : | | | tbladdr | : R3
01A9 994 : +-----+-----+-----+-----+
01A9 995 :
01A9 996 : Output Parameters:
01A9 997 :
01A9 998 : NONZERO AND result
01A9 999 :
01A9 1000 : R0 = Number of bytes remaining in the string (including the byte
01A9 1001 : that produced the NONZERO AND result)
01A9 1002 : R1 = Address of the byte that produced the NONZERO AND result
01A9 1003 : R2 = 0
01A9 1004 : R3 = tbladdr Address of 256-byte table
01A9 1005 :
01A9 1006 : AND result always ZERO (string exhausted)
01A9 1007 :
01A9 1008 : R0 = 0
01A9 1009 : R1 = Address of one byte beyond end of string
01A9 1010 : R2 = 0
01A9 1011 : R3 = tbladdr Address of 256-byte table
01A9 1012 :
01A9 1013 : Condition Codes:
01A9 1014 :
01A9 1015 : N <- 0
01A9 1016 : Z <- R0 EQL 0
01A9 1017 : V <- 0
01A9 1018 : C <- 0
01A9 1019 :
01A9 1020 : The Z bit is clear if there was a NONZERO AND result.


```
01A9 1021 : The Z bit is set if the input string is exhausted.
01A9 1022 :
01A9 1023 : Side Effects:
01A9 1024 :
01A9 1025 : This routine uses two longwords of stack.
01A9 1026 :-
01A9 1027
01A9 1028 VAX$SCANC::
50 50 3C 01A9 1029 MOVZWL R0,R0 ; Zero length string?
19 13 01AC 1030 BEQL 30$ ; Simply return if yes
5A DD 01AE 1031 PUSHL R10 ; Save R10 so it can hold handler
01B0 1032 ESTABLISH_HANDLER -
01B0 1033 STRING_ACCVIO ; Store address of condition handler
54 DD 01B5 1034 PUSHL R4 ; We need a scratch register
01B7 1035
01B7 1036 MARK_POINT SCANC_1
54 81 9A 01B7 1037 10$: MOVZBL (R1)+,R4 ; Get next character in string
01BA 1038 MARK_POINT SCANC_2
6344 52 93 01BA 1039 BITB R2,(R3)[R4] ; Index into table and AND with mask
0C 12 01BE 1040 BNEQ 40$ ; Exit loop if NONZERO
F4 50 F5 01C0 1041 SOBGTR R0,10$
01C3 1042
01C3 1043 ; If we drop through the end of the loop into the following code, then
01C3 1044 ; the input string was exhausted with no NONZERO result.
01C3 1045
0410 8F BA 01C3 1046 20$: POPR #^M<R4,R10> ; Restore saved registers
52 D4 01C7 1047 30$: CLRL R2 ; Set R2 for output value of 0
50 D5 01C9 1048 TSTL R0 ; Set condition codes
05 01CB 1049 RSB ; Return
01CC 1050
01CC 1051 ; Exit path from loop if AND produced NONZERO result
01CC 1052
51 D7 01CC 1053 40$: DECL R1 ; Point R1 to located character
F3 11 01CE 1054 BRB 20$ ; Merge with common exit
```



```
01D0 1058 .SUBTITLE VAX$SPANC - Span Characters
01D0 1059 :+
01D0 1060 : Functional Description:
01D0 1061 :
01D0 1062 : The bytes of the string specified by the length and address operands are
01D0 1063 : successively used to index into a 256 byte table whose zeroth entry
01D0 1064 : address is specified by the table address operand. The byte selected
01D0 1065 : from the table is ANDed with the mask operand. The operation continues
01D0 1066 : until the result of the AND is zero or all the bytes of the string have
01D0 1067 : been exhausted. If a zero AND result is detected, the condition code
01D0 1068 : Z-bit is cleared; otherwise, the Z-bit is set.
01D0 1069 :
01D0 1070 : Input Parameters:
01D0 1071 :
01D0 1072 : R0<15:0> = len Length of character string
01D0 1073 : R1 = addr Address of character string
01D0 1074 : R2<7:0> = mask Mask that is ANDed with successive characters
01D0 1075 : R3 = tbladdr Address of 256-byte table
01D0 1076 :
01D0 1077 : Intermediate State:
01D0 1078 :
01D0 1079 : 31 23 15 07 00
01D0 1080 : +-----+-----+-----+-----+
01D0 1081 : | delta-PC | XXXX | | len | : R0
01D0 1082 : +-----+-----+-----+-----+
01D0 1083 : | | | addr | : R1
01D0 1084 : +-----+-----+-----+-----+
01D0 1085 : | XXXXX | | XXXX | mask | : R2
01D0 1086 : +-----+-----+-----+-----+
01D0 1087 : | | | tbladdr | : R3
01D0 1088 : +-----+-----+-----+-----+
01D0 1089 :
01D0 1090 : Output Parameters:
01D0 1091 :
01D0 1092 : ZERO AND result
01D0 1093 :
01D0 1094 : R0 = Number of bytes remaining in the string (including the byte
01D0 1095 : that produced the ZERO AND result)
01D0 1096 : R1 = Address of the byte that produced the ZERO AND result
01D0 1097 : R2 = 0
01D0 1098 : R3 = tbladdr Address of 256-byte table
01D0 1099 :
01D0 1100 : AND result always NONZERO (string exhausted)
01D0 1101 :
01D0 1102 : R0 = 0
01D0 1103 : R1 = Address of one byte beyond end of string
01D0 1104 : R2 = 0
01D0 1105 : R3 = tbladdr Address of 256-byte table
01D0 1106 :
01D0 1107 : Condition Codes:
01D0 1108 :
01D0 1109 : N <- 0
01D0 1110 : Z <- R0 EQL 0
01D0 1111 : V <- 0
01D0 1112 : C <- 0
01D0 1113 :
01D0 1114 : The Z bit is clear if there was a ZERO AND result.
```



```
01D0 1115 : The Z bit is set if the input string is exhausted.
01D0 1116 :
01D0 1117 : Side Effects:
01D0 1118 :
01D0 1119 : This routine uses two longwords of stack.
01D0 1120 :-
01D0 1121
01D0 1122 VAX$SPANC::
50 50 3C 01D0 1123 MOVZWL R0,R0 ; Clear unused bits & check for 0 length
19 13 01D3 1124 BEQL 30$ ; Simply return if length is zero
5A DD 01D5 1125 PUSHL R10 ; Save R10 so it can hold handler
01D7 1126 ESTABLISH_HANDLER -
01D7 1127 STRING_ACCVIO ; Store address of condition handler
54 DD 01DC 1128 PUSHL R4 ; We need a scratch register
01DE 1129
01DE 1130 MARK_POINT SPANC_1
54 81 9A 01DE 1131 10$: MOVZBL (R1)+,R4 ; Get next character in string
01E1 1132 MARK_POINT SPANC_2
6344 52 93 01E1 1133 BITB R2,(R3)[R4] ; Index into table and AND with mask
0C 13 01E5 1134 BEQL 40$ ; Exit loop if NONZERO
F4 50 F5 01E7 1135 SOBGTR R0,10$
01EA 1136
01EA 1137 ; If we drop through the end of the loop into the following code, then
01EA 1138 ; the input string was exhausted with no ZERO result.
01EA 1139
0410 8F BA 01EA 1140 20$: POPR #^M<R4,R10> ; Restore saved registers
52 D4 01EE 1141 30$: CLRL R2 ; Set R2 for output value of 0
50 D5 01F0 1142 TSTL R0 ; Set condition codes
05 01F2 1143 RSB ; Return
01F3 1144
01F3 1145 ; Exit path from loop if AND produced ZERO result
01F3 1146
51 D7 01F3 1147 40$: DECL R1 ; Point R1 to located character
F3 11 01F5 1148 BRB 20$ ; Merge with common exit
```



```
01F7 1152 .SUBTITLE VAX$LOCC - Locate Character
01F7 1153 :+
01F7 1154 : Functional Description:
01F7 1155 :
01F7 1156 : The character operand is compared with the bytes of the string specified
01F7 1157 : by the length and address operands. Comparison continues until equality
01F7 1158 : is detected or all bytes of the string have been compared. If equality
01F7 1159 : is detected; the condition code Z-bit is cleared; otherwise the Z-bit
01F7 1160 : is set.
01F7 1161 :
01F7 1162 : Input Parameters:
01F7 1163 :
01F7 1164 : R0<15:0> = len Length of character string
01F7 1165 : R0<23:16> = char Character to be located
01F7 1166 : R1 = addr Address of character string
01F7 1167 :
01F7 1168 : Intermediate State:
01F7 1169 :
01F7 1170 : 31 23 15 07 00
01F7 1171 : +-----+-----+-----+-----+
01F7 1172 : | delta-PC | char | len | : R0
01F7 1173 : +-----+-----+-----+-----+
01F7 1174 : | addr | : R1
01F7 1175 : +-----+-----+-----+-----+
01F7 1176 :
01F7 1177 : Output Parameters:
01F7 1178 :
01F7 1179 : Character Found
01F7 1180 :
01F7 1181 : R0 = Number of bytes remaining in the string (including located one)
01F7 1182 : R1 = Address of the located byte
01F7 1183 :
01F7 1184 : Character NOT Found
01F7 1185 :
01F7 1186 : R0 = 0
01F7 1187 : R1 = Address of one byte beyond end of string
01F7 1188 :
01F7 1189 : Condition Codes:
01F7 1190 :
01F7 1191 : N <- 0
01F7 1192 : Z <- R0 EQL 0
01F7 1193 : V <- 0
01F7 1194 : C <- 0
01F7 1195 :
01F7 1196 : The Z bit is clear if the character is located.
01F7 1197 : The Z bit is set if the character is NOT located.
01F7 1198 :
01F7 1199 : Side Effects:
01F7 1200 :
01F7 1201 : This routine uses two longwords of stack.
01F7 1202 : -
01F7 1203 :
01F7 1204 VAX$LOCC::
5A DD 01F7 1205 PUSH R10 ; Save R10 so it can hold handler
01F9 1206 ESTABLISH HANDLER -
01F9 1207 STRING_ACCVIO ; Store address of condition handler
52 DD 01FE 1208 PUSH R2 ; Save register
```



```
52 50 50 F0 8F 78 0200 1209 ASHL #-16,R0,R2 ; Get character to be located
      50 50 3C 0205 1210 MOVZWL R0,R0 ; Clear unused bits & check for 0 length
      08 13 0208 1211 BEQL 20$ ; Simply return if length is 0
      020A 1212
      020A 1213 MARK_POINT LOCC_1
81 52 91 020A 1214 10$: CMPB R2,(R1)+ ; Character match?
      0A 13 020D 1215 BEQL 30$ ; Exit loop if yes
      F8 50 F5 020F 1216 SOBGTR R0,10$
      0212 1217
      0212 1218 ; If we drop through the end of the loop into the following code, then
      0212 1219 ; the input string was exhausted with the character NOT found.
      0212 1220
0404 8F BA 0212 1221 20$: POPR #^M<R2,R10> ; Restore saved R2 and R10
      50 D5 0216 1222 TSTL R0 ; Insure that C-bit is clear
      05 0218 1223 RSB ; Return with Z-bit set
      0219 1224
      0219 1225 ; Exit path when character located
      0219 1226
51 D7 0219 1227 30$: DECL R1 ; Point R1 to located character
      F5 11 021B 1228 BRB 20$ ; Join common code
```

```
021D 1232 .SUBTITLE VAX$SKPC - Skip Character
021D 1233 :+
021D 1234 : Functional Description:
021D 1235 :
021D 1236 : The character operand is compared with the bytes of the string specified
021D 1237 : by the length and address operands. Comparison continues until
021D 1238 : inequality is detected or all bytes of the string have been compared.
021D 1239 : If inequality is detected; the condition code Z-bit is cleared;
021D 1240 : otherwise the Z-bit is set.
021D 1241 :
021D 1242 : Input Parameters:
021D 1243 :
021D 1244 : R0<15:0> = len          Length of character string
021D 1245 : R0<23:16> = char       Character to be skipped
021D 1246 : R1              = addr      Address of character string
021D 1247 :
021D 1248 : Intermediate State:
021D 1249 :
021D 1250 :      31          23          15          07          00
021D 1251 : +-----+-----+-----+-----+
021D 1252 : | delta-PC | char | len | : R0
021D 1253 : +-----+-----+-----+-----+
021D 1254 : |          | addr | : R1
021D 1255 : +-----+-----+-----+-----+
021D 1256 :
021D 1257 : Output Parameters:
021D 1258 :
021D 1259 : Different Character Found
021D 1260 :
021D 1261 : R0 = Number of bytes remaining in the string (including
021D 1262 : unequal one)
021D 1263 : R1 = Address of the unequal byte
021D 1264 :
021D 1265 : All characters in string match "char"
021D 1266 :
021D 1267 : R0 = 0
021D 1268 : R1 = Address of one byte beyond end of string
021D 1269 :
021D 1270 : Condition Codes:
021D 1271 :
021D 1272 : N <- 0
021D 1273 : Z <- R0 EQL 0
021D 1274 : V <- 0
021D 1275 : C <- 0
021D 1276 :
021D 1277 : The Z bit is clear if a character different from "char" is located.
021D 1278 : The Z bit is set if the entire string is equal to "char".
021D 1279 :
021D 1280 : Side Effects:
021D 1281 :
021D 1282 : This routine uses two longwords of stack.
021D 1283 : -
021D 1284 :
021D 1285 : VAX$SKPC::
5A DD 021D 1286 : PUSH R10 ; Save R10 so it can hold handler
021F 1287 : ESTABLISH HANDLER -
021F 1288 : STRING_ACCVIO ; Store address of condition handler
```



```
52 50 50 52 DD 0224 1289      PUSHL R2          ; Save register
      8F 78 0226 1290      ASHL #16,R0,R2      ; Get character to be skipped
      50 3C 022B 1291      MOVZWL R0,R0        ; Clear unused bits & check for 0 length
      08 13 022E 1292      BEQL 20$           ; Simply return if yes
      0230 1293
      0230 1294      MARK_POINT SKPC_1
      81 52 91 0230 1295 10$: CMPB R2,(R1)+    ; Character match?
      0A 12 0233 1296      BNEQ 30$          ; Exit loop if no
      F8 50 F5 0235 1297      SOBGTR R0,10$
      0238 1298
      0238 1299 ; If we drop through the end of the loop into the following code, then
      0238 1300 ; the input string was exhausted with all of string equal to "char".
      0238 1301
      0404 8F BA 0238 1302 20$: POPR #^M<R2,R10> ; Restore saved R2 and R10
      50 D5 023C 1303      TSTL R0           ; Insure that C-bit is clear
      05 023E 1304      RSB                 ; Return with Z-bit set
      023F 1305
      023F 1306 ; Exit path when nonmatching character located
      023F 1307
      51 D7 023F 1308 30$: DECL R1           ; Point R1 to located character
      F5 11 0241 1309      BRB 20$          ; Join common code
```

```
0243 1313 .SUBTITLE VAX$MATCHC - Match Characters
0243 1314 :+
0243 1315 : Functional Description:
0243 1316 :
0243 1317 : The source string specified by the source length and source address
0243 1318 : operands is searched for a substring which matches the object string
0243 1319 : specified by the object length and object address operands. If the
0243 1320 : substring is found, the condition code Z-bit is set; otherwise, it is
0243 1321 : cleared.
0243 1322 :
0243 1323 : Input Parameters:
0243 1324 :
0243 1325 : R0<15:0> = objlen      Length of object string
0243 1326 : R1          = objaddr  Address of object string
0243 1327 : R2<15:0> = srclen     Length of source string
0243 1328 : R3          = srcaddr  Address of source string
0243 1329 :
0243 1330 : Intermediate State:
0243 1331 :
0243 1332 :      31          23          15          07          00
0243 1333 : +-----+-----+-----+-----+
0243 1334 : | delta-PC | XXXX | objlen | : R0
0243 1335 : +-----+-----+-----+-----+
0243 1336 : |          | objaddr | : R1
0243 1337 : +-----+-----+-----+-----+
0243 1338 : | XXXXX | srclen | : R2
0243 1339 : +-----+-----+-----+-----+
0243 1340 : |          | srcaddr | : R3
0243 1341 : +-----+-----+-----+-----+
0243 1342 :
0243 1343 : Output Parameters:
0243 1344 :
0243 1345 : MATCH occurred
0243 1346 :
0243 1347 : R0 = 0
0243 1348 : R1 = Address of one byte beyond end of object string
0243 1349 : R2 = Number of bytes remaining in the source string
0243 1350 : R3 = Address of one byte beyond last byte matched
0243 1351 :
0243 1352 : Strings DO NOT MATCH
0243 1353 :
0243 1354 : R0 = objlen      Length of object string
0243 1355 : R1 = objaddr     Address of object string
0243 1356 : R2 = 0
0243 1357 : R3 = Address of one byte beyond end of source string
0243 1358 :
0243 1359 : Condition Codes:
0243 1360 :
0243 1361 : N <- 0
0243 1362 : Z <- R0 EQL 0
0243 1363 : V <- 0
0243 1364 : C <- 0
0243 1365 :
0243 1366 : The Z bit is clear if the object does not match the source
0243 1367 : The Z bit is set if a MATCH occurred
0243 1368 :
0243 1369 : Side Effects:
```



```
0243 1370 :
0243 1371 :- This routine uses five longwords of stack for saved registers.
0243 1372 :-
0243 1373
0243 1374 .ENABLE LOCAL_BLOCK
0243 1375
0243 1376 VAX$MATCHC::
50 50 3C 0243 1377 MOVZWL R0,R0 ; Clear unused bits & check for 0 length
3D 13 0246 1378 BEQL 40$ ; Simply return if length is 0
52 52 3C 0248 1379 MOVZWL R2,R2 ; Clear unused bits & check for 0 length
35 13 024B 1380 BEQL 30$ ; Return with condition codes set
; based on R0 GTRU 0
5A DD 024D 1381 PUSHL R10 ; Save R10 so it can hold handler
024F 1382 ESTABLISH_HANDLER -
024F 1383 STRING_ACCVIO ; Store address of condition handler
0254 1384
0254 1385 ; The next set of instructions saves R4..R7 and copy R0..R3 to R4..R7
0254 1386
0254 1387
57 DD 0254 1388 PUSHL R7 ;
56 DD 0256 1389 PUSHL R6 ;
55 DD 0258 1390 PUSHL R5 ;
54 DD 025A 1391 PUSHL R4 ;
54 50 7D 025C 1392 MOVQ R0,R4 ;
56 52 7D 025F 1393 MOVQ R2,R6 ;
0262 1394
0A 11 0262 1395 BRB TOP_OF_LOOP ; Skip reset code on first pass
0264 1396
0264 1397 ; The following code resets the object string parameters (R0,R1) and
0264 1398 ; points the source string parameters (R2,R3) to the next byte. (Note
0264 1399 ; that there is no explicit test for R6 going to zero. That test is
0264 1400 ; implicit in the CMPL R0,R2 at TOP_OF_LOOP.)
0264 1401
0264 1402 ; In fact, this piece of code is really two nested loops. The object string
0264 1403 ; is traversed for each substring in the source string. If no match occurs,
0264 1404 ; then the source string is advanced by one character and the inner loop is
0264 1405 ; traversed again.
0264 1406
0264 1407 RESET_STRINGS:
56 D7 0264 1408 DECL R6 ; One less byte in source string
57 D6 0266 1409 INCL R7 ; ... at address one byte larger
50 54 7D 0268 1410 MOVQ R4,R0 ; Reset object string descriptor
52 56 7D 026B 1411 MOVQ R6,R2 ; Load new source string descriptor
026E 1412
026E 1413 TOP_OF_LOOP:
52 50 D1 026E 1414 CMPL R0,R2 ; Compare sizes of source and object
17 1A 0271 1415 BGTRU 50$ ; Object larger than source => NO MATCH
0273 1416 MARK_POINT MATCHC_1
83 81 91 0273 1417 10$: CMPB (R1)+,(R3)+ ; Does next character match?
EC 12 0276 1418 BNEQ RESET_STRINGS ; Exit inner loop if no match
F8 50 F5 0278 1419 SOBGTR R0,10$ ; Object exhausted?
027B 1420
027B 1421 ; If we drop through the loop, then a MATCH occurred. Set the correct
027B 1422 ; output parameters and exit. Note that R0 is equal to zero, which
027B 1423 ; will cause the condition codes (namely the Z-bit) to indicate a MATCH.
027B 1424
52 54 C2 027B 1425 SUBL R4,R2 ; Subtract objlen from srclen
027E 1426
```

```
04F0 8F BA 027E 1427 20$: POPR #M<R4,R5,R6,R7,R10> ; Restore scratch registers and R10
      0282 1428
      50 D5 0282 1429 30$: TSTL R0 ; Set condition codes
      05 0284 1430 RSB ; Return
      0285 1431
      0285 1432 ; This code executes if the object string is zero length. The upper
      0285 1433 ; 16 bits have to be cleared in R2 and then the condition codes are set
      0285 1434 ; to indicate that a MATCH occurred.
      0285 1435
      52 52 3C 0285 1436 40$: MOVZWL R2,R2 ; Clear unused bits
      F8 11 0288 1437 BRB 30$ ; Join common code
      028A 1438
      028A 1439 ; This code executes if the strings DO NOT MATCH. The actual code state
      028A 1440 ; that brings us here is that the object string is now larger than the
      028A 1441 ; remaining piece of the source string, making a match impossible.
      028A 1442
      53 57 52 D4 028A 1443 50$: CLRL R2 ; R2 contains zero in no match case
      56 C1 028C 1444 ADDL3 R6,R7,R3 ; Point R3 to end of source string
      EC 11 0290 1445 BRB 20$ ; Join common exit code
      0292 1446
      0292 1447 .DISABLE LOCAL_BLOCK
```



```
0292 1451 .SUBTITLE VAX$CRC - Calculate Cyclic Redundancy Check
0292 1452 :+
0292 1453 : Functional Description:
0292 1454 :
0292 1455 : The CRC of the data stream described by the string descriptor is
0292 1456 : calculated. The initial CRC is given by inicrc and is normally 0 or -1
0292 1457 : unless the CRC is calculated in several steps. The result is left in
0292 1458 : R0. If the polynomial is less than order-32, the result must be
0292 1459 : extracted from the result. The CRC polynomial is expressed by the
0292 1460 : contents of the 16-longword table.
0292 1461 :
0292 1462 : Input Parameters:
0292 1463 :
0292 1464 : R0 = inicrc Initial CRC
0292 1465 : R1 = tbl Address of 16-longword table
0292 1466 : R2<15:0> = strlen Length of data stream
0292 1467 : R3 = stream Address of data stream
0292 1468 :
0292 1469 : Intermediate State:
0292 1470 :
0292 1471 : 31 23 15 07 00
0292 1472 : +-----+-----+-----+-----+
0292 1473 : | | | | inicrc | : R0
0292 1474 : +-----+-----+-----+-----+
0292 1475 : | | | | tbl | : R1
0292 1476 : +-----+-----+-----+-----+
0292 1477 : | delta-PC | XXXX | strlen | : R2
0292 1478 : +-----+-----+-----+-----+
0292 1479 : | | | | stream | : R3
0292 1480 : +-----+-----+-----+-----+
0292 1481 :
0292 1482 : Output Parameters:
0292 1483 :
0292 1484 : R0 = Final CRC value
0292 1485 : R1 = 0
0292 1486 : R2 = 0
0292 1487 : R3 = Address of one byte beyond end of data stream
0292 1488 :
0292 1489 : Condition Codes:
0292 1490 :
0292 1491 : N <- R0 LSS 0
0292 1492 : Z <- R0 EQL 0
0292 1493 : V <- 0
0292 1494 : C <- 0
0292 1495 :
0292 1496 : The condition codes simply reflect the final CRC value.
0292 1497 :
0292 1498 : Side Effects:
0292 1499 :
0292 1500 : This routine uses three longwords of stack.
0292 1501 :
0292 1502 : Notes:
0292 1503 :
0292 1504 : Note that the main loop of this routine is slightly complicated
0292 1505 : by the need to allow the routine to be interrupted and restarted
0292 1506 : from its entry point. This requirement prevents R0 from being
0292 1507 : partially updated several times during each trip through the loop.
```



```
0292 1508 :      Instead, R5 is used to record the partial modifications and R5 is
0292 1509 :      copied into R0 at the last step (with the extra MOVL R5,R0).
0292 1510 :-
0292 1511
0292 1512 VAX$CRC::
52 52 3C 0292 1513      MOVZWL R2,R2      ; Clear unused bits & check for 0 length
39 13 0295 1514      BEQL 20$      ; All done if zero
5A DD 0297 1515      PUSHL R10      ; Save R10 so it can hold handler
0299 1516      ESTABLISH HANDLER -
0299 1517      STRING_ACCVIO      ; Store address of condition handler
55 55 DD 029E 1518      PUSHL R5      ; Save contents of scratch register
50 D0 02A0 1519      MOVL R0,R5      ; Copy inicrc to R5
54 DD 02A3 1520      PUSHL R4      ; Save contents of scratch register
54 D4 02A5 1521      CLRL R4      ; Clear it out (we only use R4<7:0>)
02A7 1522
02A7 1523 ; This is the main loop that operates on each byte in the input stream
02A7 1524
02A7 1525      MARK_POINT CRC_1
55 83 8C 02A7 1526 10$: XORB2 (R3)+,R5      ; Include next byte
02AA 1527
02AA 1528 ; The next three instructions are really the body of a loop that executes
02AA 1529 ; twice on each pass through the outer loop. Rather than incur additional
02AA 1530 ; overhead, this inner loop is expanded in line.
02AA 1531
54 55 55 F0 8F 8B 02AA 1532      BICB3 #^XF0,R5,R4      ; Get right 4 bits
55 55 1C 04 EF 02AF 1533      EXTZV #4,#28,R5,R5      ; Shift result right 4
02B4 1534      MARK_POINT CRC_2
55 6144 CC 02B4 1535      XORL2 (R1)[R4],R5      ; Include table entry
02B8 1536
54 55 55 F0 8F 8B 02B8 1537      BICB3 #^XF0,R5,R4      ; Get right 4 bits
55 55 1C 04 EF 02BD 1538      EXTZV #4,#28,R5,R5      ; Shift result right 4
02C2 1539      MARK_POINT CRC_3
55 6144 CC 02C2 1540      XORL2 (R1)[R4],R5      ; Include table entry
02C6 1541
50 55 D0 02C6 1542      MOVL R5,R0      ; Preserve latest complete result
02C9 1543
DB 52 F5 02C9 1544      SOBGTR R2,10$      ; Count down loop
02CC 1545
0430 8F BA 02CC 1546      POPR #^M<R4,R5,R10>      ; Restore saved R4, R5, and R10
02D0 1547
51 D4 02D0 1548 20$: CLRL R1      ; R1 must be zero on exit
50 D5 02D2 1549      TSTL R0      ; Determine N- and Z-bits
02D4 1550      ; (Note that TSTL clears V- and C-bits)
05 02D4 1551      RSB      ; Return to caller
```



```
02D5 1555 .SUBTITLE STRING_ACCVIO - Exception Dispatcher
02D5 1556 :+
02D5 1557 : Functional Description:
02D5 1558 :
02D5 1559 : This routine receives control when an access violation occurs while
02D5 1560 : executing within the emulator. This routine determines whether the
02D5 1561 : exception occurred while accessing a source or destination string.
02D5 1562 : (This check is made based on the PC of the exception.)
02D5 1563 :
02D5 1564 : If the PC is one that is recognized by this routine, then the state of
02D5 1565 : the instruction (character counts, string addresses, and the like) are
02D5 1566 : restored to a state where the instruction/routine can be restarted
02D5 1567 : after the cause for the exception is eliminated. Control is then
02D5 1568 : passed to a common routine that sets up the stack and the exception
02D5 1569 : parameters in such a way that the instruction or routine can restart
02D5 1570 : transparently.
02D5 1571 :
02D5 1572 : If the exception occurs at some unrecognized PC, then the exception is
02D5 1573 : reflected to the user as an exception that occurred within the
02D5 1574 : emulator.
02D5 1575 :
02D5 1576 : There are two exceptions that can occur that are not backed up to a
02D5 1577 : consistent state.
02D5 1578 :
02D5 1579 : 1. If stack overflow occurs due to use of the stack by one of
02D5 1580 : the VAX$xxxxxx routines, it is unlikely that this routine
02D5 1581 : will even execute because the code that transfers control
02D5 1582 : here must first copy the parameters to the exception stack
02D5 1583 : and that operation would fail. (The failure causes control
02D5 1584 : to be transferred to VMS, where the stack expansion logic is
02D5 1585 : invoked and the routine resumed transparently.)
02D5 1586 :
02D5 1587 : 2. If assumptions about the address space change out from under
02D5 1588 : these routines (because an AST deleted a portion of the
02D5 1589 : address space or a similar silly thing), the handling of the
02D5 1590 : exception is UNPREDICTABLE.
02D5 1591 :
02D5 1592 : Input Parameters:
02D5 1593 :
02D5 1594 : R0 - Value of SP when the exception occurred
02D5 1595 : R1 - PC of exception
02D5 1596 : R2 - Scratch
02D5 1597 : R3 - Scratch
02D5 1598 : R10 - Address of this routine (but that was already used so R10
02D5 1599 : can be used for a scratch register if needed)
02D5 1600 :
02D5 1601 : 00(SP) - Saved R0 (Contents of R0 when exception occurred)
02D5 1602 : 04(SP) - Saved R1 (Contents of R1 when exception occurred)
02D5 1603 : 08(SP) - Saved R2 (Contents of R2 when exception occurred)
02D5 1604 : 12(SP) - Saved R3 (Contents of R3 when exception occurred)
02D5 1605 :
02D5 1606 : 16(SP) - Return PC in exception dispatcher in operating system
02D5 1607 :
02D5 1608 : 20(SP) - First longword of system-specific exception data
02D5 1609 : xx(SP) - First longword of system-specific exception data
02D5 1610 :
02D5 1611 : The address of the next longword is the position of the stack when
```


02D5 1612 : the exception occurred. This address is contained in R0 on entry
02D5 1613 : to this routine.
02D5 1614 :
02D5 1615 : R0 -> <4*<N+1> + 16>(SP) - Instruction-specific data
02D5 1616 : . - Optional instruction-specific data
02D5 1617 : - Saved R10
02D5 1618 : <4*<N+M> + 16>(SP) - Return PC from VAX\$xxxxxx routine (M is the number
02D5 1619 : of instruction-specific longwords, including the
02D5 1620 : saved R10. M is guaranteed greater than zero.)
02D5 1621 :
02D5 1622 : Implicit Input:
02D5 1623 :
02D5 1624 : It is assumed that the contents of all registers (except R0 to R3)
02D5 1625 : coming into this routine are unchanged from their contents when the
02D5 1626 : exception occurred. (For R0 through R3, this assumption applies to the
02D5 1627 : saved register contents on the top of the stack. Any modification to
02D5 1628 : these registers must be made to their saved copies and not to the
02D5 1629 : registers themselves.)
02D5 1630 :
02D5 1631 : It is further assumed that the exception PC is within the bounds of
02D5 1632 : this module. (Violation of this assumption is simply an inefficiency.)
02D5 1633 :
02D5 1634 : Finally, the macro BEGIN_MARK_POINT should have been invoked at the
02D5 1635 : beginning of this module to define the symbols
02D5 1636 :
02D5 1637 : MODULE BASE
02D5 1638 : PC_TABLE_BASE
02D5 1639 : HANDLER_TABLE_BASE
02D5 1640 : TABLE_SIZE
02D5 1641 :
02D5 1642 : Output Parameters:
02D5 1643 :
02D5 1644 : If the exception is recognized (that is, if the exception PC is
02D5 1645 : associated with one of the mark points), control is passed to the
02D5 1646 : context-specific routine that restores the instruction state to a
02D5 1647 : uniform point from which it can be restarted.
02D5 1648 :
02D5 1649 : R0 - Value of SP when exception occurred
02D5 1650 : R1 - scratch
02D5 1651 : R2 - scratch
02D5 1652 : R3 - scratch
02D5 1653 : R10 - scratch
02D5 1654 :
02D5 1655 : R0 -> zz(SP) - Instruction-specific data begins here
02D5 1656 :
02D5 1657 : The instruction-specific routines eventually pass control back to the
02D5 1658 : host system with the following register contents.
02D5 1659 :
02D5 1660 : R0 - Address of return PC from VAX\$xxxxxx routine
02D5 1661 : R1 - Byte offset from top of stack (into saved R0 through R3)
02D5 1662 : to indicate where to store the delta-PC (if so required)
02D5 1663 : R10 - Restored to its value on entry to VAX\$xxxxxx
02D5 1664 :
02D5 1665 : If the exception PC occurred somewhere else (such as a stack access),
02D5 1666 : the saved registers are restored and control is passed back to the
02D5 1667 : host system with an RSB instruction.
02D5 1668 :


```
02D5 1669 ; Implicit Output:
02D5 1670 ;
02D5 1671 ; The register contents are modified to put the intermediate state of
02D5 1672 ; the instruction into a consistent state from which it can be
02D5 1673 ; continued. Any changes to R0 through R3 are made in their saved state
02D5 1674 ; on the top of the stack. Any scratch registers saved by each
02D5 1675 ; VAX$xxxxxx routine are restored.
02D5 1676 ; -
02D5 1677 ;
02D5 1678 STRING_ACCVIO:
FD25 52 D4 02D5 1679 CLRL R2 ; Initialize the counter
51 CF 9F 02D7 1680 PUSHAB MODULE_BASE ; Store base address of this module
51 8E C2 02DB 1681 SUBL2 (SP)+,R1 ; Get PC relative to this base
0000'CF42 51 B1 02DE 1682
07 13 02DE 1683 10$: CMPW R1,PC_TABLE_BASE[R2] ; Is this the right PC?
F4 52 17 F2 02E4 1684 BEQL 20$ ; Exit loop if true
02E6 1685 AOBLS #TABLE_SIZE,R2,10$ ; Do the entire table
02EA 1686
02EA 1687 ; If we drop through the dispatching based on PC, then the exception is not
02EA 1688 ; one that we want to back up. We simply reflect the exception to the user.
02EA 1689
OF BA 02EA 1690 POPR #^M<R0,R1,R2,R3> ; Restore saved registers
05 05 02EC 1691 RSB ; Let VMS reflect the exception
02ED 1692
02ED 1693 ; The exception PC matched one of the entries in our PC table. R2 contains
02ED 1694 ; the index into both the PC table and the handler table. R1 has served
02ED 1695 ; its purpose and can be used as a scratch register.
02ED 1696
51 0000'CF42 3C 02ED 1697 20$: MOVZWL HANDLER_TABLE_BASE[R2],R1 ; Get the offset to the handler
FD08 CF41 17 02F3 1698 JMP MODULE_BASE[RT] ; Pass control to the handler
```

```
02F8 1700      .SUBTITLE      Packing Routines for String Instructions
02F8 1701      :+
02F8 1702      : Functional Description:
02F8 1703      :
02F8 1704      :   These routines are used to store the intermediate state of the state
02F8 1705      :   of the string instructions (except MOVTC and MOVTUC) into the registers
02F8 1706      :   that are altered by a given instruction.
02F8 1707      :
02F8 1708      : Input Parameters:
02F8 1709      :
02F8 1710      :   R0 - Points to top of stack when exception occurred
02F8 1711      :
02F8 1712      :   See each routine- and context-specific entry point for more details.
02F8 1713      :
02F8 1714      :   In general, register contents for counters and string pointers that
02F8 1715      :   are naturally tracking through a string are not listed. Register
02F8 1716      :   contents that are out of the ordinary (different from those listed
02F8 1717      :   in the intermediate state pictures in each routine header) are listed.
02F8 1718      :
02F8 1719      : Output Parameters:
02F8 1720      :
02F8 1721      :   R0 - Points to return PC from VAX$xxxxxx
02F8 1722      :   R1 - Locates specific byte in R0..R3 that will contain the delta-PC
02F8 1723      :
02F8 1724      :   All scratch registers (including R10) that are not supposed to be
02F8 1725      :   altered by the routine are restored to their contents when the
02F8 1726      :   routine was originally entered.
02F8 1727      :
02F8 1728      : Notes:
02F8 1729      :
02F8 1730      :   In all of the instruction-specific routines, the state of the stack
02F8 1731      :   will be shown as it was when the exception occurred. All offsets will
02F8 1732      :   be pictured relative to R0. In addition, relevant contents of R0
02F8 1733      :   through R3 will be listed as located in the registers themselves, even
02F8 1734      :   though the actual code will manipulate the saved values of these
02F8 1735      :   registers located on the top of the stack.
02F8 1736      :
02F8 1737      :   The apparent arbitrary order of the instruction-specific routines is
02F8 1738      :   dictated by the amount of code that they can share. The most sharing
02F8 1739      :   occurs at the middle of the code, for instructions like CMPC5 and
02F8 1740      :   SCANC. The CRC routines, because they are the only routines that store
02F8 1741      :   the delta-PC in R2 appear first. The CMPC3 instruction has no
02F8 1742      :   instruction-specific code that cannot be shared with all of the other
02F8 1743      :   routines so it appears at the end.
02F8 1744      : -
02F8 1745      :
02F8 1746      : .ENABLE      LOCAL_BLOCK
02F8 1747      :
02F8 1748      : +
02F8 1749      : CRC Packing Routine
02F8 1750      :
02F8 1751      :   R4 - Scratch
02F8 1752      :   R5 - Scratch
02F8 1753      :
02F8 1754      :   00(R0) - Saved R4
02F8 1755      :   04(R0) - Saved R5
02F8 1756      :   08(R0) - Saved R10
```



```
02F8 1757 : 12(R0) - Return PC
02F8 1758 :
02F8 1759 : If entry is at CRC_2 or CRC_3, the exception occurred after the string
02F8 1760 : pointer, R3, was advanced. That pointer must be backed up to achieve a
02F8 1761 : consistent state.
02F8 1762 :-
02F8 1763 :-
02F8 1764 CRC_2:
02F8 1765 CRC_3:
OC AE D7 02F8 1766 DECL PACK_L_SAVED_R3(SP) ; Back up string pointer
54 80 7D 02FB 1767 CRC_1: MOVQ (R0)+,R4 ; Restore R4 and R5
51 0B 9A 02FE 1768 MOVZBL #CRC_B_DELTA_PC,R1 ; Indicate offset used to store delta-PC
29 11 0301 1770 BRB 30$ ; Not much common code left but use it
0303 1771 :-
0303 1772 :+ MATCHC Packing Routine
0303 1773 :
0303 1774 : R4<15:0> - Number of characters in object string
0303 1775 : R5 - Address of object string
0303 1776 : R6<15:0> - Number of characters remaining in source string
0303 1777 : R7 - Updated pointer into source string
0303 1778 :
0303 1779 : 00(R0) - Saved R4
0303 1780 : 04(R0) - Saved R5
0303 1781 : 08(R0) - Saved R6
0303 1782 : 12(R0) - Saved R7
0303 1783 : 16(R0) - Saved R10
0303 1784 : 20(R0) - Return PC
0303 1785 :
0303 1786 :
0303 1787 : Note that the MATCHC instruction is backed up to the top of its inner loop.
0303 1788 : That is, when the instruction restarts, it will begin looking for a match
0303 1789 : between the first character of the object string and the latest starting
0303 1790 : character in the source string.
0303 1791 :-
0303 1792 :-
0303 1793 MATCHC_1:
08 6E 54 7D 0303 1794 MOVQ R4,PACK_L_SAVED_R0(SP) ; Reset object string to its beginning
AE 56 7D 0306 1795 MOVQ R6,PACK_L_SAVED_R2(SP) ; Reset to updated start of source string
54 80 7D 030A 1796 MOVQ (R0)+,R4 ; Restore R4 and R5
56 80 7D 030D 1797 MOVQ (R0)+,R6 ; ... and R6 and R7
17 11 0310 1798 BRB 20$ ; Exit through common code path
0312 1799 :-
0312 1800 :+ CMPC5 Packing Routine
0312 1801 :
0312 1802 : R4<7:0> - Fill character operand
0312 1803 :
0312 1804 : 00(R0) - Saved R4
0312 1805 : 04(R0) - Saved R10
0312 1806 : 08(R0) - Return PC
0312 1807 :
0312 1808 :-
0312 1809 :-
0312 1810 CMPC5_1:
0312 1811 CMPC5_2:
02 AE 54 90 0312 1812 CMPC5_3:
0312 1813 MOV B R4,CMPC5_B_FILL(SP) ; Pack "fill" into R0<23:16>
```

```
03 11 0316 1814 BRB 10$ ; Merge with code to restore R4
      0318 1815
      0318 1816 ;+
      0318 1817 : SCANC and SPANC Packing Routine
      0318 1818 :
      0318 1819 : R4 - Scratch
      0318 1820 :
      0318 1821 : 00(R0) - Saved R4
      0318 1822 : 04(R0) - Saved R10
      0318 1823 : 08(R0) - Return PC
      0318 1824 :
      0318 1825 : If entry is at SCANC_2 or SPANC_2, the exception occurred after the string
      0318 1826 : pointer, R1, was advanced. That pointer must be backed up to achieve a
      0318 1827 : consistent state.
      0318 1828 : -
      0318 1829 :
      0318 1830 SCANC_2:
      0318 1831 SPANC_2:
04 AE D7 0318 1832 DECL PACK_L_SAVED_R1(SP) ; Back up string pointer
      0318 1833 SCANC_1:
      0318 1834 SPANC_1:
54 80 D0 0318 1835 10$: MOVL (R0)+,R4 ; Restore R4
      031E 1836 BRB 20$ ; Exit through common code path
      0320 1837
      0320 1838 ;+
      0320 1839 : LOCC and SKPC Packing Routine
      0320 1840 :
      0320 1841 : R2<7:0> - Character operand
      0320 1842 :
      0320 1843 : 00(R0) - Saved R2
      0320 1844 : 04(R0) - Saved R10
      0320 1845 : 08(R0) - Return PC
      0320 1846 : -
      0320 1847 :
      0320 1848 LOCC_1:
      0320 1849 SKPC_1:
      0320 1850
      0320 1851 ASSUME LOCC_B_CHAR EQ SKPC_B_CHAR
      0320 1852
02 AE 08 AE 90 0320 1853 MOVB PACK_L_SAVED_R2(SP),LOCC_B_CHAR(SP) ; Pack 'char' into R0<23:16>
      08 AE 80 D0 0325 1854 MOVL (R0)+,PACK_L_SAVED_R2(SP) ; Restore saved R2
      0329 1855
      0329 1856 ;+
      0329 1857 : CMPC3 Packing Routine
      0329 1858 :
      0329 1859 : 00(R0) - Saved R10
      0329 1860 : 04(R0) - Return PC
      0329 1861 : -
      0329 1862 :
      0329 1863 ASSUME CMPC5_B_DELTA_PC EQ CMPC3_B_DELTA_PC
      0329 1864 ASSUME SCANC_B_DELTA_PC EQ CMPC3_B_DELTA_PC
      0329 1865 ASSUME SPANC_B_DELTA_PC EQ CMPC3_B_DELTA_PC
      0329 1866 ASSUME LOCC_B_DELTA_PC EQ CMPC3_B_DELTA_PC
      0329 1867 ASSUME SKPC_B_DELTA_PC EQ CMPC3_B_DELTA_PC
      0329 1868 ASSUME MATCHC_B_DELTA_PC EQ CMPC3_B_DELTA_PC
      0329 1869
      0329 1870 CMPC3_1:
```


51	03	9A	0329	1871	20\$:	MOVZBL	#CMPC3 B DELTA_PC,R1	:	Indicate that R0 gets delta PC
5A	80	D0	032C	1872	30\$:	MOVL	(R0)+,R10	:	Restore saved R10
			032F	1873					
			032F	1874		ASSUME	PACK_V_FPD LE 15	:	Insure that both of these bits
			032F	1875		ASSUME	PACK_V_FPD LE 15	:	can be contained in a word
			032F	1876					
51	0300	8F	A8	032F	1877	BISW	#<PACK_M_FPD!-	:	Indicate that FPD gets set
				0334	1878		PACK_M_ACCVIO>,R1	:	Exception is an access violation
	FCC9'	31	0334	1879		BRW	VAX\$REFLECT_FAULT	:	Modify stack and reflect exception
			0337	1880					
			0337	1881		.DISABLE	LOCAL_BLOCK		

```
0337 1883 .SUBTITLE Packing Routines for MOVTC and MOVTUC
0337 1884 :+
0337 1885 : Functional Description:
0337 1886 :
0337 1887 : These routines are used to store the intermediate state of the state
0337 1888 : of the MOVTC and MOVTUC instructions into the registers R0 through R5.
0337 1889 : The main reason for keeping these two routines separate from the rest
0337 1890 : of the string instructions is that R10 is not stored directly adjacent
0337 1891 : to the return PC. This means that there is no code that can be shared
0337 1892 : with the rest of the instructions.
0337 1893 :
0337 1894 : Input Parameters:
0337 1895 :
0337 1896 : R0 - Points to top of stack when exception occurred
0337 1897 :
0337 1898 : See the context-specific entry point for more details.
0337 1899 :
0337 1900 : Output Parameters:
0337 1901 :
0337 1902 : R0 - Points to return PC from VAX$xxxxxx
0337 1903 : R1 - Locates specific byte in R0..R3 that will contain the delta-PC
0337 1904 :
0337 1905 : All scratch registers (including R10) that are not supposed to be
0337 1906 : altered by the routine are restored to their contents when the
0337 1907 : routine was originally entered.
0337 1908 :
0337 1909 : Notes:
0337 1910 :
0337 1911 : See the notes in the routine header for the storage routines for
0337 1912 : the rest of the string instructions.
0337 1913 :-
```


0337 1915 :+
0337 1916 : MOVTC Packing Routine (if moving in the FORWARD direction)
0337 1917 :
0337 1918 : The entry points MOVTC_1, MOVTC_2, and MOVTC_3 are used when moving the
0337 1919 : string in the forward direction. If the entry is at MOVTC_2, then the
0337 1920 : source and destination strings are out of synch and R1 must be adjusted
0337 1921 : (decremented) to keep the two strings in step.
0337 1922 :
0337 1923 : In the MOVE_FORWARD routine, there is a need for a scratch register before
0337 1924 : the fill character is used. R2 is used as this scratch and its original
0337 1925 : contents, the fill character, are saved on the stack. The entry points
0337 1926 : MOVTC_1 and MOVTC_2 have the stack in this state.
0337 1927 :
0337 1928 : R2 - Scratch
0337 1929 :
0337 1930 : 00(R0) - Saved R2
0337 1931 : 04(R0) - Saved R10
0337 1932 : 08(R0) - Saved R0
0337 1933 : <31:16> - Initial contents of R0
0337 1934 : <15:00> - Contents of R0 at time of latest entry to VAX\$MOVTC
0337 1935 : 12(R0) - Saved R4
0337 1936 : <31:16> - Initial contents of R4
0337 1937 : <15:00> - Contents of R4 at time of latest entry to VAX\$MOVTC
0337 1938 : 16(R0) - Return PC
0337 1939 :
0337 1940 : If entry is at MOVTC_3, then there are no registers other than R0 and R4
0337 1941 : (and of course R10) that are saved on the stack.
0337 1942 :
0337 1943 : 00(R0) - Saved R10
0337 1944 : 04(R0) - Saved R0
0337 1945 : <31:16> - Initial contents of R0
0337 1946 : <15:00> - Contents of R0 at time of latest entry to VAX\$MOVTC
0337 1947 : 08(R0) - Saved R4
0337 1948 : <31:16> - Initial contents of R4
0337 1949 : <15:00> - Contents of R4 at time of latest entry to VAX\$MOVTC
0337 1950 : 12(R0) - Return PC
0337 1951 :
0337 1952 : The following are register contents at the time that the exception occurred.
0337 1953 :
0337 1954 : R0 - Number of bytes remaining to be modified in source string
0337 1955 : R1 - Address of current byte in source string (except at MOVTC_2)
0337 1956 : R2 - Junk or fill character (if entry at MOVTC_3)
0337 1957 : R3 - Address of translation table (unchanged during execution)
0337 1958 : R4 - Signed difference between current lengths of source and destination
0337 1959 : R5 - Address of current byte in destination string
0337 1960 :
0337 1961 : R10 - Access violation handler address (so can be used as scratch)
0337 1962 :
0337 1963 : Note that if R4 LSSU 0, then the value of R0 represents the number of bytes
0337 1964 : in the source string remaining to be modified. There are also excess bytes
0337 1965 : of the source string that will be untouched by the complete execution of
0337 1966 : this instruction. (In fact, at completion, R0 will contain the number of
0337 1967 : unmodified bytes.)
0337 1968 :
0337 1969 : Note further that entry at MOVTC_3 is impossible with R4 LSSU 0 because
0337 1970 : MOVTC_3 indicates that an access violation occurred while storing the
0337 1971 : fill character in the destination and that can only happen when the output

```
0337 1972 : string is longer than the input string.
0337 1973 :
0337 1974 : The state that must be modified before being stored depends on the sign of
0337 1975 : R4, which in turn depends on which of source and destination is longer.
0337 1976 :
0337 1977 : R4 GEQU 0 => srclen LEQU dstlen
0337 1978 :
0337 1979 : R0 - unchanged
0337 1980 : R4 - increased by R0 (R4 <- R4 + R0)
0337 1981 :
0337 1982 : R4 LSSU 0 => srclen GTRU dstlen
0337 1983 :
0337 1984 : R0 - increased by negative of R4 (R0 <- R0 + ABS(R4))
0337 1985 : R4 - replaced with input value of R0 (R4 <- R0)
0337 1986 :-
0337 1987 :
0337 1988 : .ENABLE LOCAL_BLOCK
0337 1989 :
0337 1990 MOVTC_2:
0337 1991 DECL PACK_L_SAVED_R1(SP) ; Back up source string
0337 1992 :
0337 1993 MOVTC_1:
0337 1994 MOVL (R0)+,PACK_L_SAVED_R2(SP) ; Restore contents of saved R2
0337 1995 TSTL R4 ; R4 LSSU 0 => srclen GTRU dstlen
0337 1996 BGEQ 5$ ; Branch if srclen LEQU dstlen
0337 1997 MNEGL R4,R10 ; Save absolute value of difference
0337 1998 MOVL PACK_L_SAVED_R0(SP),R4 ; Get updated dstlen (R4 <- R0)
0337 1999 ADDL R10,PACK_L_SAVED_R0(SP) ; ... and updated srclen (R0 <- R0 - R4)
0337 2000 BRB 10$
0337 2001 :
0337 2002 5$: ADDL PACK_L_SAVED_R0(SP),R4 ; Reset correct count of destination
0337 2003 :
0337 2004 MOVTC_3:
0337 2005 10$: MOVL (R0)+,R10 ; Restore saved R10
0337 2006 :
0337 2007 ASSUME MOVTC_W_INISRLEN EQ MOVTC_W_INISRLEN
0337 2008 :
0337 2009 MOVW 2(R0),MOVTC_W_INISRLEN(SP) ; Save high-order word of R0
0337 2010 ADDL #4,R0 ; Point R0 to saved R4
0337 2011 MOVW R4,(R0) ; Store low order R4 in saved R4
0337 2012 MOVL (R0)+,R4 ; Restore all of R4
0337 2013 :
0337 2014 ASSUME MOVTC_B_DELTA_PC EQ MOVTC_B_DELTA_PC
0337 2015 :
0337 2016 : Indicate that R2<31:24> gets delta-PC and cause the FPD bit to be set
0337 2017 :
0337 2018 MOVL #<MOVTC_B_DELTA_PC!- ; Locate delta-PC offset
0337 2019 PACK_M_FPD!- ; Set FPD bit in exception PSL
0337 2020 PACK_M_ACCVIO>,R1 ; Indicate an access violation
0337 2021 :
0337 2022 ASSUME MOVTC_M_FPD EQ MOVTC_M_FPD
0337 2023 ASSUME MOVTC_B_FLAGS EQ MOVTC_B_FLAGS
0337 2024 :
0337 2025 BISB #MOVTC_M_FPD,MOVTC_B_FLAGS(SP) ; Set internal FPD bit
0337 2026 BRW VAX$REFLECT_FAULT ; Reflect exception to user
```

04 AE D7

08 AE 80 D0

5A 54 0B 18

5A 54 CE

54 6E D0

6E 5A C0

03 11

54 6E C0

5A 80 D0

02 AE 02 A0 B0

50 04 C0

60 54 B0

54 80 D0

51 0000030B 8F D0

09 AE 01 88

FC91' 31


```
036F 2028 :+
036F 2029 : MOVTC Packing Routine (if moving in the BACKWARD direction)
036F 2030 :
036F 2031 : The entry points MOVTC_4, MOVTC_5, and MOVTC_6 are used when moving the
036F 2032 : string in the backward direction. If the entry is at MOVTC_6, then the
036F 2033 : source and destination strings are out of synch and R1 must be adjusted
036F 2034 : (incremented) to keep the two strings in step.
036F 2035 :
036F 2036 : At entry points MOVTC_5 and MOVTC_6, we must reset the source string
036F 2037 : pointer, R1, to the beginning of the string because it is currently set up
036F 2038 : to traverse the string from its high-address end. The details of this reset
036F 2039 : operation depend on the relative lengths of the source and destination
036F 2040 : strings as described below.
036F 2041 :
036F 2042 : At all three entry points, we must reset the destination string
036F 2043 : pointer, R5, to the beginning of the string because it is currently
036F 2044 : set up to traverse the string from its high-address end.
036F 2045 :
036F 2046 :     00(R0) - Saved R10
036F 2047 :     04(R0) - Saved R0
036F 2048 :             <31:16> - Initial contents of R0
036F 2049 :             <15:00> - Contents of R0 at time of latest entry to VAX$MOVTC
036F 2050 :     08(R0) - Saved R4
036F 2051 :             <31:16> - Initial contents of R4
036F 2052 :             <15:00> - Contents of R4 at time of latest entry to VAX$MOVTC
036F 2053 :     12(R0) - Return PC
036F 2054 :
036F 2055 : The following are register contents at the time that the exception occurred.
036F 2056 :
036F 2057 :     R0 - Number of bytes remaining to be modified in source string
036F 2058 :     R1 - Address of current byte in source string (except at MOVTC_6)
036F 2059 :     R2 - scratch
036F 2060 :     R3 - Address of translation table (unchanged during execution)
036F 2061 :     R4 - Signed difference between current lengths of source and destination
036F 2062 :     R5 - Address of current byte in destination string
036F 2063 :
036F 2064 :     R10 - Access violation handler address (so can be used as scratch)
036F 2065 :
036F 2066 : Note that if R4 LSSU 0, then the value of R0 represents the number of bytes
036F 2067 : in the source string remaining to be modified. There are also excess bytes
036F 2068 : of the source string that will be untouched by the complete execution of
036F 2069 : this instruction. (In fact, at completion, R0 will contain the number of
036F 2070 : unmodified bytes.)
036F 2071 :
036F 2072 : Note further that entry at MOVTC_4 is impossible with R4 LSSU 0 because
036F 2073 : MOVTC_4 indicates that an access violation occurred while storing the
036F 2074 : fill character in the destination and that can only happen when the output
036F 2075 : string is longer than the input string.
036F 2076 :
036F 2077 : The state that must be modified before being stored depends on the sign of
036F 2078 : R4, which in turn depends on which of source and destination is longer.
036F 2079 :
036F 2080 :     R4 GEQU 0 => srclen LEQU dstlen
036F 2081 :
036F 2082 :         R0 - unchanged
036F 2083 :         R1 - backed up by R0 (R1 <- R1 - R0)
036F 2084 :         R4 - increased by R0 (R4 <- R4 + R0)
```

```
036F 2085 : R5 - backed up by new value of R4 (R5 <- R5 - R4)
036F 2086 :
036F 2087 : R4 LSSU 0 => srclen GTRU dstlen
036F 2088 :
036F 2089 : R0 - increased by negative of R4 (R0 <- R0 + ABS(R4))
036F 2090 : R1 - backed up by input value of R0 (R1 <- R1 - R0)
036F 2091 : R4 - replaced with input value of R0 (R4 <- R0)
036F 2092 : R5 - backed up by new value of R4 (R5 <- R5 - R4)
036F 2093 :
036F 2094 : Note that R1 is modified before R0 is changed
036F 2095 :-
036F 2096 :
036F 2097 MOVTC_6:
04 AE D6 036F 2098 INCL PACK_L_SAVED_R1(SP) ; Undo last fetch from source string
0372 2099
0372 2100 MOVTC_5:
04 AE 6E C2 0372 2101 SUBL PACK_L_SAVED_R0(SP),PACK_L_SAVED_R1(SP)
0376 2102 ; Point R1 to start of source string
54 D5 0376 2103 TSTL R4 ; R4 LSSU 0 => srclen GTRU dstlen
0B 18 0378 2104 BGEQ 20$ ; Branch if srclen LEQU dstlen
5A 54 CE 037A 2105 MNEGL R4,R10 ; Save absolute value of difference
54 6E D0 037D 2106 MOVL PACK_L_SAVED_R0(SP),R4 ; Get updated dstlen (R4 <- R0)
6E 5A C0 0380 2107 ADDL R10,PACK_L_SAVED_R0(SP) ; ... and updated srclen (R0 <- R0 - R4)
03 11 0383 2108 BRB 30$
0385 2109
0385 2110 MOVTC_4:
54 6E C0 0385 2111 20$: ADDL PACK_L_SAVED_R0(SP),R4 ; Treat two strings as having same length
55 54 C2 0388 2112 30$: SUBL R4,R5 ; Point R5 to start of destination string
C3 11 038B 2113 BRB 10$ ; Join common code
```



```
038D 2115 :+
038D 2116 : MOVTUC Packing Routine
038D 2117 :
038D 2118 : Note that R7 is used to count the number of remaining characters in the
038D 2119 : strings. The other two counts, R0 and R4, are set to contain their final
038D 2120 : values.
038D 2121 :
038D 2122 : If R0 was initially smaller than R4,
038D 2123 :
038D 2124 :     R0 - 0
038D 2125 :     R4 - Difference between R4 and R0 (R4-R0)
038D 2126 :     R7 - Number of characters remaining in source (shorter) string
038D 2127 :
038D 2128 : If R0 was initially larger than R4,
038D 2129 :
038D 2130 :     R0 - Difference between R0 and R4 (R0-R4)
038D 2131 :     R4 - 0
038D 2132 :     R7 - Number of characters remaining in destination (shorter) string
038D 2133 :
038D 2134 : In either case, the stack when the exception occurred looks like this.
038D 2135 :
038D 2136 :     R6 - Scratch
038D 2137 :     R7 - Number of characters remaining in two strings
038D 2138 :
038D 2139 :     00(R0) - Saved R6
038D 2140 :     04(R0) - Saved R7
038D 2141 :     08(R0) - Saved R10
038D 2142 :     12(R0) - Saved R0
038D 2143 :             <31:16> - Initial contents of R0
038D 2144 :             <15:00> - Contents of R0 at time of latest entry to VAX$MOVTUC
038D 2145 :     16(R0) - Saved R4
038D 2146 :             <31:16> - Initial contents of R4
038D 2147 :             <15:00> - Contents of R4 at time of latest entry to VAX$MOVTUC
038D 2148 :     20(R0) - Return PC
038D 2149 :
038D 2150 : If the entry is at MOVTUC_2 or MOVTUC_3, then the source and
038D 2151 : destination strings are out of synch and R1 must be adjusted
038D 2152 : (decremented) to keep the two strings in step.
038D 2153 : -
038D 2154 :
038D 2155 : MOVTUC_2:
038D 2156 : MOVTUC_3:
038D 2157 :     DECL    PACK_L_SAVED_R1(SP)    ; Back up source string pointer
0390 2158 :
0390 2159 : MOVTUC_1:
0390 2160 :     ADDL    R7,PACK_L_SAVED_R0(SP) ; Readjust source string count
0393 2161 :     ADDL    R7,R4                  ; ... and destination string count
0396 2162 :     MOVQ    (R0)+,R6                ; Restore saved R6 and R7
0399 2163 :     BRB     10$                     ; Join exit path shared with MOVTUC
039B 2164 :
039B 2165 :     .DISABLE    LOCAL_BLOCK
039B 2166 :
039B 2168 :     END_MARK_POINT
039B 2169 :
039B 2170 :     .END
```

04 AE D7

6E 57 C0
54 57 C0
56 80 7D
B5 11

VAX\$STRING
Symbol table

VAX-11 Character String Instruction Emul 16-SEP-1984 01:30:09 VAX/VMS Macro V04-00 Page 46
7-SEP-1984 17:13:25 [EMULAT.SRC]VAXSTRING.MAR;2 (20)

```

...PC = 000002C2
CMPC3_1 = 00000329 R 02
CMPC3_B_DELTA_PC = 00000003
CMPC5_1 = 00000312 R 02
CMPC5_2 = 00000312 R 02
CMPC5_3 = 00000312 R 02
CMPC5_B_DELTA_PC = 00000003
CMPC5_B_FILL = 00000002
CRC_1 = 000002FB R 02
CRC_2 = 000002FB R 02
CRC_3 = 000002FB R 02
CRC_B_DELTA_PC = 0000000B
ESCAPE = 0000010A R 02
HANDLER_TABLE_BASE = 00000000 R 04
IDENTICAL = 00000176 R 02
LOCC_1 = 00000320 R 02
LOCC_B_CHAR = 00000002
LOCC_B_DELTA_PC = 00000003
MATCHC_1 = 00000303 R 02
MATCHC_B_DELTA_PC = 00000003
MODULE_BASE = 00000000 R 02
MODULE_END = 0000039B R 02
MOVE_BACKWARD = 00000063 R 02
MOVE_FORWARD = 00000027 R 02
MOVTC_1 = 0000033A R 02
MOVTC_2 = 00000337 R 02
MOVTC_3 = 00000350 R 02
MOVTC_4 = 00000385 R 02
MOVTC_5 = 00000372 R 02
MOVTC_6 = 0000036F R 02
MOVTC_B_DELTA_PC = 0000000B
MOVTC_B_FLAGS = 00000009
MOVTC_M_FPD = 00000001
MOVTC_V_FPD = 00000000
MOVTC_W_INISRLEN = 00000002
MOVTUC_1 = 00000390 R 02
MOVTUC_2 = 0000038D R 02
MOVTUC_3 = 0000038D R 02
MOVTUC_B_DELTA_PC = 0000000B
MOVTUC_B_FLAGS = 00000009
MOVTUC_M_FPD = 00000001
MOVTUC_V_FPD = 00000000
MOVTUC_W_INISRLEN = 00000002
NO_MATCH = 000001A4 R 02
PACK_L_SAVED_R0 = 00000000
PACK_L_SAVED_R1 = 00000004
PACK_L_SAVED_R2 = 00000008
PACK_L_SAVED_R3 = 0000000C
PACK_M_ACCVIO = 00000200
PACK_M_FPD = 00000100
PACK_V_FPD = 00000008
PC_TABLE_BASE = 00000000 R 03
PSE$M_V = 00000002
RESET_STRINGS = 00000264 R 02
SCANC_1 = 0000031B R 02
SCANC_2 = 00000318 R 02
SCANC_B_DELTA_PC = 00000003

```

```

SKPC_1 = 00000320 R 02
SKPC_B_CHAR = 00000002
SKPC_B_DELTA_PC = 00000003
SPANC_1 = 0000031B R 02
SPANC_2 = 00000318 R 02
SPANC_B_DELTA_PC = 00000003
STRING_ACCVIO = 000002D5 R 02
TABLE_SIZE = 00000017
TOP_OF_LOOP = 0000026E R 02
VAX$CMPC3 = 0000012C RG 02
VAX$CMPC5 = 00000152 RG 02
VAX$CRC = 00000292 RG 02
VAX$LOCC = 000001F7 RG 02
VAX$MATCHC = 00000243 RG 02
VAX$MOVTC = 00000000 RG 02
VAX$MOVTUC = 000000A0 RG 02
VAX$REFLECT_FAULT = ***** X 00
VAX$SCANC = 000001A9 RG 02
VAX$SKPC = 0000021D RG 02
VAX$SPANC = 000001D0 RG 02

```


+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR
\$AB\$\$	00000000 (0.)	01 (1.)	NOPIC USR
VAX\$CODE	0000039B (923.)	02 (2.)	PIC USR
PC TABLE	0000002E (46.)	03 (3.)	PIC USR
HANDLER_TABLE	0000002E (46.)	04 (4.)	PIC USR

	CON	ABS	LCL	NOSHR	NOEXE	NORD	NOWRT	NOVEC	BYTE
	CON	ABS	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE
	CON	REL	LCL	SHR	EXE	RD	NOWRT	NOVEC	LONG
	CON	REL	LCL	SHR	NOEXE	RD	NOWRT	NOVEC	BYTE
	CON	REL	LCL	SHR	NOEXE	RD	NOWRT	NOVEC	BYTE

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	16	00:00:00.05	00:00:01.23
Command processing	76	00:00:00.46	00:00:04.46
Pass 1	189	00:00:05.84	00:00:21.67
Symbol table sort	0	00:00:00.25	00:00:00.77
Pass 2	370	00:00:04.05	00:00:15.02
Symbol table output	9	00:00:00.07	00:00:00.07
Psect synopsis output	2	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	662	00:00:10.75	00:00:43.25

The working set limit was 1500 pages.
41149 bytes (81 pages) of virtual memory were used to buffer the intermediate code.
There were 20 pages of symbol table space allocated to hold 197 non-local and 56 local symbols.
2170 source lines were read in Pass 1, producing 20 object records in Pass 2.
26 pages of virtual memory were used to define 24 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
\$255\$DUA28:[EMULAT.OBJ]VAXMACROS.MLB;1	15
\$255\$DUA28:[SYSLIB]STARLET.MLB;2	5
TOTALS (all libraries)	20

301 GETS were required to define 20 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:VAXSTRING/OBJ=OBJ\$:VAXSTRING MSRC\$:VAXSTRING/UPDATE=(ENH\$:VAXSTRING)+LIB\$:VAXMACROS/LIB

0145 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

